Bachelor-Thesis

Augmented Reality Visualisierung medizinischer Bilddaten auf mobilen Geräten

Universität Heidelberg, Hochschule Heilbronn

Studiengang Medizinische Informatik

Vorgelegt von:	Tamara Wiebe
Geboren am:	08. März 1990
Matrikelnummer:	176110
Referent:	Prof. Dr. Rolf Bendl
Korreferentin:	PD Dr. Lena Maier-Hein
Betreuer des DKFZ:	Dipl. Inform. (FH) Thomas Kilgus
Eingereicht am:	3. März 2014

Kurzfassung

Augmented Reality Anwendungen können in der Medizin Eingriffe erleichtern, beispielsweise durch intraoperative Projektion von Zugangswegen oder Tumoren und Risikostrukturen auf den Patienten. Gründe, weshalb die Verwendung von Augmented Reality noch keinen Einzug in den Operationssaal gefunden hat, sind unter anderem eine erschwerte Tiefenwahrnehmung der virtuellen Objekte in der echten Szene und fehlende Möglichkeiten, um die Fülle der zusätzlich visualisierten Objekte zu kontrollieren. Diesen Hindernissen entgegenzutreten ist das Ziel dieser Arbeit. Weiterhin gehört eine echtzeitfähige Implementierung zu den Anforderungen dieser Arbeit, um die Visualisierung im Rahmen eines Projektes zu nutzen, welches Augmented Reality auf mobilen Geräten direkt am Patienten zeigt. Um diese Ziele zu erreichen, wurde zunächst eine Texturprojektion kombiniert mit einem selbstentwickelten Grafikkartenprogramm realisiert, um dem Betrachter die Orientierung innerhalb der Augmented Reality Szene zu erleichtern und die Berechnung der perspektivischen Projektion der Textur zugleich effizient zu halten. Um die Tiefenwahrnehmung in der Szene zu verbessern, wurde ein weiteres Grafikkartenprogramm entwickelt, welches in eine gegebene Oberfläche eine Öffnung zeichnet, durch welche der Betrachter in das Innere des Patienten blicken kann. Weiterhin wurde ein Konzept umgesetzt, mit dessen Hilfe die Anzahl an abgebildeten Objekten in der Augmented Reality Szene gesteuert werden kann. Dieses Konzept dient außerdem der Untergliederung von Objekten in verschiedene Familien, für die dann unterschiedliche Darstellungen umgesetzt werden können. Ergebnis ist sowohl eine sichtbar verbesserte Tiefenwahrnehmung als auch ein Konzept zur Kontrolle der Fülle an abgebildeten Informationen in einer echtzeitfähigen Implementierung.

Danksagung

Zunächst möchte ich mich bei Herrn Prof. Dr. Hans-Peter Meinzer bedanken, der es mir ermöglicht hat, diese Arbeit in der Medizinischen und Biologischen Informatik am Deutschen Krebsforschungszentrum unter solch angenehmen Arbeitsbedingungen zu schreiben. Der gesamten Abteilung möchte ich für die enorme Hilfsbereitschaft bei Fragen jeglicher Art danken. Großer Dank gebührt der Leiterin der Juniorgruppe Computer Assisted Interventions, PD Dr. Lena Maier-Hein, für die stetige Unterstützung, Ratschläge und Verbesserungstipps, und nicht zuletzt für das interessante Thema dieser Arbeit. Meinem Referenten Herrn Prof. Dr. Bendl danke ich für die Korrektur, Betreuung und Bewertung dieser Arbeit und für die Hilfsbereitschaft zu jeder Zeit. Besonderer Dank geht an meinen Betreuer Thomas Kilgus, der stets ein offenes Ohr für Fragen meinerseits hatte, für das Korrekturlesen, für Verbesserungstipps und für die investierte Zeit und Geduld in die Betreuung dieser Arbeit. Weiterhin möchte ich Markus Fangerau für die Einführung in die Computergrafik, die damit verbundene Unterstützung bei der Implementierung und das Korrekturlesen danken, insbesondere für die Zeit, die dafür investiert wurde.

Meinen Eltern danke ich für die jahrelange Unterstützung in jeglicher Hinsicht, ohne euch wäre das Studium so nicht machbar gewesen! Angela und meiner Schwester Cindy danke ich für die stets aufmunternden Worte. Meiner Tante Monika sei für die vielen Abende bei ihr gedankt, die mir oft zum Abschalten verhalfen.

Großer Dank gebührt nicht zuletzt meinem Freund Marcus Schleppe. Für die ständige und geduldige Unterstützung und auch für die aufbauenden Worte, wenn mal nichts zu klappen schien, kann ich nur dankbar sein.

I. INHALTSVERZEICHNIS

I. Inhaltsverzeichnis 1
1 Einleitung 3
1.1 Motivation 4
1.2 Zielsetzung 5
1.3 Aufbau der Arbeit 5
2 Grundlagen 7
2.1 Computergrafik 7
2.1.1 Texturmapping 7
2.1.2 Die Open Graphics Library 8
2.2 Die Klassenbibliothek VTK 12
2.2.1 Die VTK Rendering Pipeline 12
2.2.2 Shader in VTK 13
2.3 Das Softwareframework MITK 13
2.4 Akquisition von Tiefendaten 15
2.5 Konzept der mobilen Augmented Reality 16
3 Verwandte Arbeiten 19
4 Methoden 23
4.1 Ansatz 23
4.2 Umsetzung 24
4.2.1 Texturprojektion 26
4.2.2 Shader zur Optimierung der Texturprojektion 31
4.2.3 Shader zur Berechnung einer Öffnung in der Hautoberfläche 40
4.2.4 Benutzerselektion relevanter Strukturen 42
5 Ergebnisse 44
5.1 Fokus-und-Kontext-Konzept 44
5.2 Texturprojektion 47

5.3 Projektion einer Öffnung in die Hautoberfläche des Patienten 49
5.4 Kombination der Konzepte 50
6 Diskussion & Ausblick 53
7 Referenzen 56
II. Anhang 60
A Abkürzungsverzeichnis 60
B Abbildungsverzeichnis 61
C Tabellenverzeichnis 65
D Ausgewählte Implementierungen 66
E DVD 70
III. Eidesstattliche Erklärung 71

1 EINLEITUNG

Statistiken zufolge gehört Krebs zu den zehn häufigsten Todesursachen in Deutschland.[1] Deshalb ist die Erforschung und Optimierung der Behandlungsmöglichkeiten von bösartigen Wucherungen ein zentraler Bestandteil der Forschung. Neben der Bestrahlung und der medikamentösen Behandlung stellt die Resektion eine Therapiemöglichkeit von Tumoren dar. Bei dieser Form der Behandlung stehen die Entfernung des pathologischen Gewebes und zugleich die Schonung von gesundem Gewebe im Mittelpunkt. Zur Vorbeugung von Komplikationen, beispielsweise eine mangelhafte oder falsche Planung der Resektion und der Zugangswege, werden heutzutage vermehrt computergestützte Planungssysteme eingesetzt, die aus zuvor akquirierten Bilddaten optimale Resektionsvorschläge errechnen können. Allerdings stellt eine sorgfältige Planung zwar eine notwendige, aber keine hinreichende Bedingung für den Erfolg des chirurgischen Eingriffs dar. Die mentale Übertragung der zweidimensionalen Planungsdaten, dargestellt auf einem Bildschirm, auf die reale dreidimensionale Anatomie des Patienten erfordert ein hohes Vorstellungsvermögen. [2] Deshalb bedarf es einer Integration der Planungsdaten in den operativen Workflow, um die Gefahr von Komplikationen während der Operation, wie die Verletzung von umliegenden Gefäßen, Nerven oder Organstrukturen, zu verringern. [3] Eine Synthese der Planungsdaten mit der realen Szene bietet eine Möglichkeit, dem operierenden Chirurg das Resektionsvolumen zusammen mit der individuellen Patientenanatomie zu veranschaulichen. Die Synthese von computergenerierten Objekten mit realen Bildern wird als Augmented Reality bezeichnet. [4] Abgesehen von der Tumorchirurgie gibt es viele weitere medizinische Einsatzgebiete für Augmented Reality. Beispielhaft seien hier minimalinvasive Eingriffe im Allgemeinen genannt, die aufgrund des geringen Traumas weniger belastend für den Patienten sind, die aber wegen des kleinen Eintrittskanals eine Sicht in das Innere des Patienten unmöglich machen. Auch hier kann die Projektion präoperativer Daten in das Sichtfeld des Arztes dessen Orientierung erleichtern. [5] Somit entwickelten sich in den letzten Jahren viele Ansätze, um den Einzug von Augmented Reality in den Operationssaal zu ermöglichen. Jedoch stellt die wirklichkeitsnahe und räumlich korrekte Darstellung solcher zusätzlicher Informa-

tionen eine große Herausforderung dar, weshalb der Gebrauch von Augmented Reality noch nicht zum medizinischen Alltag gehört.

1.1 MOTIVATION

Obwohl die Verwendung von Augmented Reality in der Medizin viele Vorteile für den operierenden Arzt bietet, gilt das Verfahren bisher nicht als Standardverfahren zur Unterstützung der Planung und Durchführung von Operationen. Dies lässt sich auf verschiedene Nachteile der bisherigen Visualisierung zurückführen.

Zunächst werden durch das Einblenden virtueller Informationen zwangsläufig Komponenten der realen Szene überdeckt. Bisher gibt es wenige Ansätze, die die Bestandteile der realen Szene so mit den künstlichen Objekten kombinieren, dass zum einen das Verständnis der Szene erhalten bleibt und zum anderen der Fokus des Betrachters auf ein bestimmtes Objekt gelenkt wird. Dies hat zur Folge, dass Augmented Reality Bilder oft überladen wirken und der Betrachter mit der Fülle an Informationen überfordert wird. [30]

Zusätzlich ist eine Projektion eines dreidimensionalen Objekts (virtuelles Objekt) auf eine zweidimensionale Ebene (reale Szene als Bild) mit Informationsverlust hinsichtlich der räumlichen Lage der Objekte verbunden. Somit mangelt es den Szenen an Tiefeninformation und der Betrachter kann nur schwer herausfinden, wo genau das virtuelle Objekt in Wirklichkeit liegt. In den momentan verfügbaren Augmented Reality Anwendungen wirken virtuelle Objekte demnach so als würden sie oberhalb der realen Szene „schweben“ (vgl. Abbildung 1). Dies erschwert dem Benutzer die Vorstellung von der Lage der virtuellen Objekte im Kontext der echten Szene.

Abbildung 1: Bisherige Augmented Reality Visualisierung mit geringer Tiefenwahrnehmung (siehe Tumor)

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, die Darstellung der Augmented Reality im Kontext der mobilen markerlosen Augmented Reality (vgl. Kapitel 2.5) zu verbessern. Um die Verwendung von Augmented Reality im medizinischen Umfeld zu ermöglichen, muss insbesondere die Visualisierung der künstlichen Objekte verbessert werden. Verschiedene Anwendungsgebiete erfordern verschiedene Visualisierungstechniken, weshalb diese Thesis darauf abzielt, Möglichkeiten für unterschiedliche Einsatzbedingungen umzusetzen, die gut miteinander kombiniert werden können. Hierbei sollen insbesondere die im obigen Abschnitt erläuterten Probleme vermindert werden. Die zentralen Ziele dieser Arbeit sind:

- Verbesserung der Tiefenwahrnehmung
- Echtzeitfähige Implementierung der Visualisierung (Eigenschaft von Augmented Reality Systemen im Allgemeinen [6])
- Kontrolle der Fülle an abgebildeten Informationen und optische Gruppierung der Objekte

1.3 Aufbau der Arbeit

In Kapitel 2 werden die für diese Arbeit relevanten Grundlagen erläutert. Hier wird das benötigte Wissen bezüglich Software (The Medical Interaction Toolkit, The Visualization Toolkit, Computergrafikprogrammierung) vermittelt. Außerdem werden zwei Technologien zur Akquisition von Tiefenbilddaten dargelegt. Zuletzt wird das Konzept der mobilen Augmented Reality erklärt und in diesem Zusammenhang zwei bestehende Methoden zur Umsetzung aufgezeigt.

Kapitel 3 behandelt die zu dieser Thesis verwandten Arbeiten. Es werden verschiedene Ansätze vorgestellt, die sich mit den in Abschnitt 1.1 genannten Problemen bei der Augmented Reality Visualisierung auseinandersetzen.

In Abschnitt 4 wird zunächst der Ansatz zur Erreichung der oben genannten Ziele erläutert und anschließend die Umsetzung der Ziele beschrieben.

In Kapitel 5 werden die Ergebnisse dieser Arbeit veranschaulicht. In diesem Zuge werden insbesondere Vorteile der entwickelten Visualisierungskonzepte gegenüber der vorherigen Augmented Reality Darstellung gezeigt.

Abschließend werden in Kapitel 6 die Ergebnisse diskutiert und ein Ausblick auf zukünftige Arbeiten zu diesem Thema gegeben.

2 GRUNDLAGEN

Im Folgenden werden die Grundlagen erläutert, die zum Verständnis der Arbeit nötig sind. Zunächst werden wichtige Konzepte der Computergrafik erörtert, um die in der Thesis entstandenen Grafikkartenprogramme nachvollziehen zu können. Ein Abriss aus dem Visualization Toolkit und eine Erklärung des Aufbaus des Medical Interaction Toolkit vermitteln das nötige Wissen hinsichtlich der Entwicklungsumgebung dieser Arbeit. Weiterhin werden zwei Möglichkeiten zur Akquisition von Tiefenbilddaten und deren Funktion im Medizinbereich dargelegt, welche den Grundbaustein für das Verständnis des darauffolgenden Grundlagenkapitels legen. Dieses abschließende Kapitel behandelt den Begriff der mobilen Augmented Reality, in deren Kontext die Augmented Reality Visualisierung in dieser Thesis verbessert werden soll.

2.1 COMPUTERGRAFIK

Der folgende Abschnitt gibt einen Überblick über in dieser Arbeit verwendete Konzepte der Computergrafik. Es werden die Grundlagen zum Texturmapping vermittelt und anschließend eine Einführung in die OpenGraphics Library (OpenGL) und in die zugehörigen Shading Language (GLSL) gegeben.

2.1.1 TEXTURMAPPING

Der Begriff Texturmapping beschreibt die Technik, ein bestimmtes Muster auf ein Objekt passgenau zu projizieren. Im Allgemeinen beschreibt eine Textur eine beliebige Art von Muster. Eine Textur kann also sowohl ein künstlich generiertes Muster als auch ein Farbbild sein. Die Elemente einer Textur, sogenannte Texel, sind in Texturkoordinaten definiert. Um eine Textur perspektivisch korrekt auf ein Objekt abzubilden, ist es notwendig, für jeden Punkt des Objekts die zugehörigen Texturkoordinaten zu ermitteln. Mathematisch lässt sich der Gedanke wie folgt ausdrücken: T(s,t) sei ein zweidimensionales Texturmuster mit Texturkoordinaten s,t. Die Punkte des geometrischen Objekts sind als dreidimensionale Koordinaten (x,y,z) angegeben. Dann erfüllt die Texture Map hinsichtlich der Objektkoordinaten (x,y,z) und der Texturkoordinaten (s,t) folgende Gleichungen:

$$x = x(s,t), \quad (1)$$

$$y = y(s,t), \quad (2)$$

$$z = z(s,t). \quad (3)$$

(siehe [12])

2.1.2 Die Open Graphics Library

OpenGL ist eine hardwareunabhängige Grafikprogrammierschnittstelle. Sie stellt Möglichkeiten zur Verfügung, um 3D-Modelle mithilfe von geometrischen Primitiven, wie Punkte, Linien und Polygone, zu erstellen und diese Modelle durch verschiedene grafische Effekte wirklichkeitsgetreu zu gestalten. [14] Im Folgenden werden zunächst wichtige Konzepte der Fixed Function Rendering Pipeline von OpenGL und in diesem Kontext der Zweck verschiedener Koordinatentransformationen verdeutlicht. Anschließend werden die für diese Thesis relevanten Konstrukte der OpenGL Shading Language erläutert.

- **Die Fixed Function Rendering Pipeline**

Die Fixed Function Rendering Pipeline beschreibt die sequenzielle Abarbeitung mehrerer Rechenschritte, um mithilfe geometrischer Primitiven und Pixel-Daten Objekte mit unterschiedlichen grafischen Effekten zu visualisieren. Zu den geometrischen Primitiven zählen Polygone, die aus mehreren verbundenen Vertices bestehen, und somit dreidimensionale Modelle in Objektkoordinaten repräsentieren. Ein Vertex beschreibt einen Punkt in einem dreidimensionalen Raum. Vertex-Operationen sind nun dafür zuständig, auf einen Vertex verschiedene Transformationen anzuwenden, um sie von Objektkoordinaten auf Bildschirmkoordinaten zu übertragen. Abbildung 2 gibt einen Überblick über die nötigen Schritte, um Objektkoordinaten in Bildschirmkoordinaten zu überführen.

Abbildung 2: Überblick über durchzuführende Transformationen, um Objektkoordinaten in Bildschirmkoordinaten zu überführen. Die Modell- und Augenpunktstransformation (rotes Rechteck) dient zur Berechnung der Weltkoordinaten. Eine Projektionstransformation (gelbes Rechteck) legt das sichtbare Volumen (Frustum) der Szene fest. Nach einer Normierung werden mithilfe einer Viewport-Transformation (grünes Rechteck) die Bildschirmkoordinaten berechnet. Nach: [13]

Zunächst werden die Modelle mithilfe einer Modell- und einer Augenpunktstransformation (rotes Rechteck in Abbildung 2) in das Weltkoordinatensystem übertragen. Dieses Weltkoordinatensystem ist dasjenige, welches OpenGL benutzt, um die gesamte Szene zu beschreiben. Die Transformation positioniert das Objekt in der Gesamtszene und legt außerdem den Punkt fest, von welchem die Szene betrachtet wird. Über eine Projektionstransformation (gelbes Rechteck in Abbildung 2) wird das sichtbare Volumen, genannt Frustum, in Projektionskoordinaten definiert. Dadurch werden alle Vertices, die außerhalb des Frustums liegen, entfernt. Nach einer Normierung der Koordinaten werden mithilfe der Viewport-Transformation (grünes Rechteck in Abbildung 2) die Projektionskoordinaten in Bildschirmkoordinaten umgerechnet. [13]

Weiterhin werden Normalenvektoren, Texturkoordinaten und Farbwerte der Vertices generiert. Durch eine lineare Interpolation dieser Komponenten von drei Vertices werden die Farbwerte der Pixel im Bildschirm berechnet, die das von den Vertices erzeugte Polygon bedeckt. Durch Fragment-Operationen können dann bestimmte farbliche Effekte, beispielsweise aus Texturen, visualisiert werden. Ein Fragment entspricht einem Pixel im Bildschirmspeicher, welche wiederrum einem Rasterpunkt des Bildschirms entsprechen. [15]

- **Die OpenGL Shading Language**

Ein Teil der oben genannten Vertex- und Fragment-Operationen können in der Fixed Function Rendering Pipeline durch eine eigene Implementierung ersetzt werden. Diese Programme nennen sich Shader, wobei mithilfe eines Vertex-Shaders Vertex-Operationen und mittels eines Fragment-Shaders Fragment-Operationen durchgeführt werden können. Hierfür gibt es eine eigene Sprache, die OpenGL Shading Language (GLSL). Auf einige wichtige Konstrukte dieser

Sprache, die in dieser Arbeit zur Shaderprogrammierung verwendet wurden, wird im Folgenden eingegangen.

Tabelle 1 gibt einen Überblick über die in dieser Arbeit verwendeten Parameter eines Shaderprogrammes und deren Funktionen.

Parameter	Name	Funktion/Zweck
Eingabeparameter für den Vertex-Shader	`gl_Vertex`	Objektkoordinate des Vertex
	`gl_Normal`	Normale des Vertex
	`gl_MultiTexCoord0`	Texturkoordinate des Vertex
Uniform Attribute	-	Attribute, die aus der Anwendung, die den Shader aufruft, an den Shader übergeben werden
Ausgabeparameter des Vertex-Shaders	`gl_Position`	Projektionskoordinate des Vertex
Varying Variablen	-	Zur Übergabe von Werten aus dem Vertex- an den Fragment-Shader
Eingebaute varying Variablen	`gl_TexCoord[0]`	Texturkoordinate des Vertex zur Übergabe an den Fragment-Shader
Ausgabeparameter des Fragment-Shaders	`gl_FragColor`	finale Farbe des Fragments

Tabelle 1: Überblick über wichtige Attribute und Variablen eines Vertex- und eines Fragment-Shaders. Nach [16]

Mithilfe Abbildung 3 werden die in Tabelle 1 genannten Shaderparameter im Folgenden genauer erläutert.

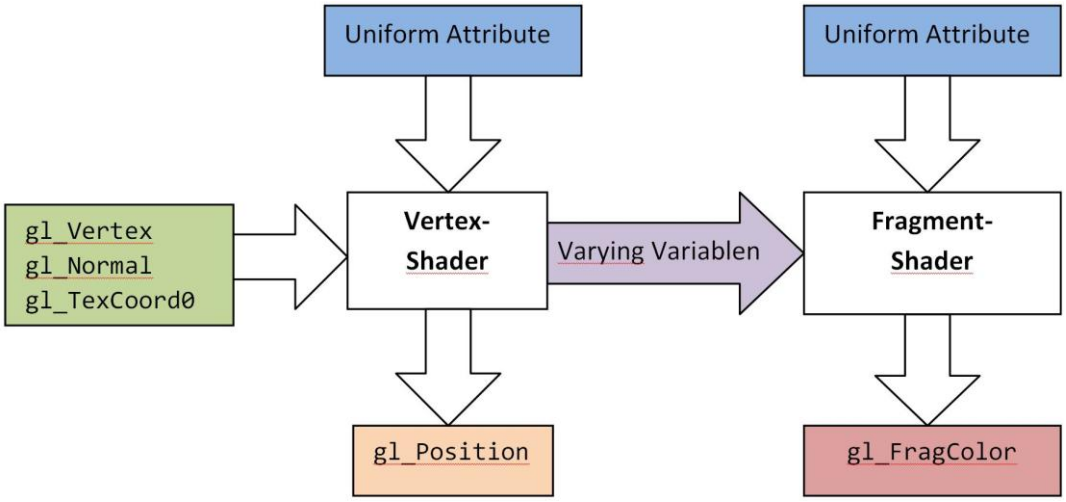

Abbildung 3: Ablauf der Bearbeitung eines Eingangsvertex durch einen Vertex- und einen Fragmentshader. Die Eingabewerte (grünes Rechteck) des Vertex und Werte aus der Anwendung, die den Shader aufruft ("`uniform`", blaues Rechteck), werden im Vertex-Shader verarbeitet. Die transformierte Position des Vertex (`gl_Position`, oranges Rechteck) wird zur Weiterverarbeitung in der Pipeline ausgegeben. Als Eingabeparameter erhält der Fragment-Shader verschiedene eingebaute und benutzerdefinierte Attribute aus dem Vertex-Shader ("`varying`, lila Pfeil) und Werte aus der Anwendung ("`uniform`", blaues Rechteck), die den Shader aufruft. Als Ausgabeparameter wird die endgültige Farbe eines Fragments (`gl_FragColor`, rotes Rechteck) in den Bildschirmspeicher geschrieben. Nach [16]

Als Eingabewerte (Abbildung 3, grünes Rechteck) erhält der Vertex-Shader die Koordinate des Vertex, die zugehörige Normale und die Texturkoordinate des Vertex. Aus der Anwendung, die den Shader aufruft, können ebenfalls Attribute ("uniform", Abbildung 3, blaues Rechteck links) als Eingangsparameter gesetzt werden. Nach der Ausführung des Vertex-Shaders wird die in Projektionskoordinaten transformierte Position des Vertex (`gl_Position`, Abbildung 3, oranges Rechteck) ausgegeben. Die Eingabewerte des Fragment-Shaders setzen sich aus den uniform-Attributen aus der Anwendung (Abbildung 3, blaues Rechteck rechts) und sogenannten varying Variablen zusammen. Varying Variablen repräsentieren Werte aus dem Vertex-Shader, die in den Fragment-Shader propagiert werden und müssen im Vertex-Shader explizit als `varying` deklariert werden. GLSL bietet hierfür schon eingebaute Variablen an, beispielsweise `gl_TexCoord[0]`, die für die Übergabe der Texturkoordinate des Vertex an den Fragment-Shader zuständig ist. Nach der Ausführung des Fragment-Shaders wird die endgültige Farbe eines Fragments mithilfe der Ausgabevariable `gl_FragColor` in den Bildschirmspeicher geschrieben. [16]

2.2 Die Klassenbibliothek VTK

VTK[1] ist eine open source Software für dreidimensionale Computergrafik, Bildverarbeitung und Visualisierung, in welcher verschiedene Visualisierungsalgorithmen implementiert sind. Der folgende Abschnitt gibt eine kurze Einführung in die Rendering Pipeline und die Shaderintegration in VTK. Das in dieser Arbeit verwendete Softwareframework MITK übernimmt und erweitert diese Elemente, weshalb die Kenntnis der zugrunde liegenden Ideen wichtig für das Verständnis der entwickelten Lösungsansätze dieser Thesis ist.

2.2.1 Die VTK Rendering Pipeline

Das zentrale Element von VTK ist die VTK Rendering Pipeline. Diese stellt eine sequenzielle Abarbeitung von Schritten dar, um Eingangsdaten so zu verarbeiten, dass sie auf dem Bildschirm dargestellt werden können.

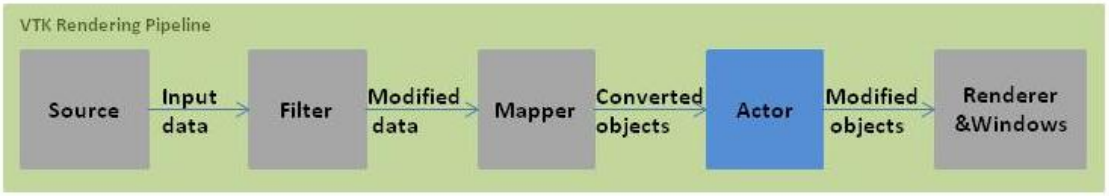

Abbildung 4: VTK Rendering Pipeline: Eingabedaten (Source) werden zunächst gefiltert (Filter) und zu visualisierbaren Objekten konvertiert (Mapper). Diese Objekte nennen sich Actor (blau). Einem Actor werden verschiedene Visualisierungsparameter hinzugefügt, bis schließlich die perspektivische Darstellung auf dem Bildschirm vom Renderer und zugehörigen RenderWindows realisiert wird. Nach [17]

Eingangsdaten ("Source" in Abbildung 4) können entweder von einer Datei (beispielsweise von einer Bilddatei) eingelesen oder synthetisch generiert werden. Diese Daten werden von einem oder mehreren Filtern modifiziert und an den Mapper übergeben. Der Mapper stellt die Schnittstelle zwischen der Verarbeitung der Eingangsdaten (Source und Filter) und der Bearbeitung zur Darstellung auf dem Bildschirm (Actor und Renderer & Windows) her. Er erstellt aus den Eingabedaten Objekte, die auf dem Bildschirm von der Rendering Engine dargestellt werden können. Diese Objekte nennen sich Actors, die jeweils für sich verschiedene Visualisierungsparameter (beispielsweise Transparenz und Farbe) beinhalten können. Die endgültige Visualisierung übernimmt ein sogenannter Renderer, der für die perspektivische Darstellung auf dem Bildschirm

[1] http://www.vtk.org/

zuständig ist. Das zugehörige Renderwindow stellt sozusagen die Leinwand für einen Renderer dar, auf welche er die Objekte zeichnet.

Jeder Actor (blaues Rechteck in Abbildung 4) repräsentiert ein sichtbares Objekt in einer Szene. Jeder Actor besitzt eine Liste von Eigenschaften, sogenannter Properties. Eine Instanz einer Property speichert beispielsweise die Farbe, eine andere die Transparenz eines Actors. [17]

2.2.2 Shader in VTK

Für eine komfortable Verwendung von Hardwareshadern wurde in VTK eine Schnittstelle zu verschiedenen Shading Languages (insbesondere auch zur GLSL, die für die Shaderprogrammierung in dieser Arbeit verwendet wurde) hergestellt. Ein Shaderprogramm realisiert bestimmte grafische Effekte für Vertices und Fragmente eines Objekts, also eines Actors in VTK. Deshalb soll für jeden Actor ein Shaderprogramm individuell angewendet werden können. Die Verknüpfung eines Actors mit einem Shaderprogramm wird durch eine Instanz einer Property dieses Actors hergestellt. Über diese Instanz muss zunächst das Shaderprogramm als xml-Datei geladen werden, was durch einen einfachen Funktionsaufruf mit Übergabe des Dateipfades des Shaders möglich ist. Die Instanz bietet außerdem eine Funktion an, um Eingabeparameter des Shaderprogramms (entspricht einem als `uniform` deklarierten Attribut im Shader, vgl. Abschnitt 2.1.2) zu setzen. Die Aufnahme des Shaders in die Rendering Pipeline wird von VTK übernommen. [18]

2.3 Das Softwareframework MITK

Das Softwareframework MITK ist ein open source Softwaresystem zur Entwicklung von Bildverarbeitungsanwendungen in der Medizin. Im Folgenden wird nun der Aufbau dieses Frameworks erläutert, welches als Entwicklungsumgebung für die Implementierung dieser Arbeit dient.

Anfänglich stellte das MITK ein Toolkit dar, welches die beiden Frameworks „The Insight Toolkit“ (ITK) und VTK (vgl. Abschnitt 2.2) erweitert und kombiniert [7]. In diesem Zuge wurde die VTK Rendering Pipeline in MITK integriert und weiterentwickelt, um an dieses geradlinige Konzept der Visualisierung anzuknüpfen.

Aus der Idee einer Kombination von ITK und VTK entwickelte sich eine modularisierte Entwicklungsumgebung. Es gibt sogenannte Module, wobei ein Modul eine bestimmte Problemdomäne abdeckt. In großen Software-Systemen werden häufig Singletons - global erreichbare statische Instanzen - genutzt, um zentrale Aufgaben in der gesamten Applikation zu erfüllen. MITK bietet für die Realisierung solcher Aufgaben die sogenannten C++ Micro Services[2] an. Micro Services sind C++ Objekte, die an einer zentralen Stelle dynamisch registriert werden können, um der Applikation bestimmte Funktionen bereit zu stellen. Mithilfe des CommonToolkit Plugin Frameworks wird das Hinzufügen oder Entfernen verschiedener Komponenten und des zugehörigen Codes zur Laufzeit ermöglicht. Diese Komponenten werden in diesem Zusammenhang Bundles oder auch Plugins genannt. [8]

Des Weiteren wurden in MITK Möglichkeiten entwickelt, Daten, die mit verschiedenen Modalitäten (Ultraschall, Time-Of-Flight Kamera) aufgenommen wurden, zu verwalten und weiterzuverarbeiten. Im Kontext dieser Arbeit ist insbesondere die Time-Of-Flight (ToF) Kamera-Integration in MITK zu erwähnen, welche die Einbindung von Tiefenbildkameras ermöglicht. Abbildung 5 gibt einen Überblick über die Struktur des Moduls MITK-ToF.

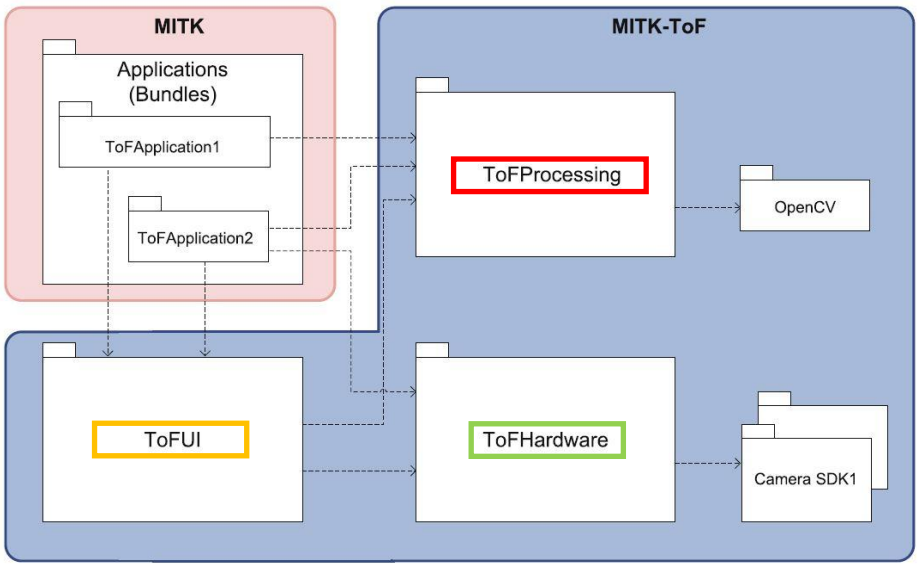

Abbildung 5: Struktur des Moduls MITK-ToF, bestehend aus drei Schichten: Die Hardwareschicht ToFHardware (grün), die Verarbeitungsschickt ToFProcessing (rot) und die Anwendungsschicht ToFUI (gelb). Diese Schichten stellen die Schnittstelle zur Kamerahardware (ToFHardware) und zum Benutzer (ToFUI) her und sind weiterhin zuständig für die Verarbeitung der gewonnenen Bilddaten (ToFProcessing). MITK-ToF kann beliebig von verschiedenen Bundles genutzt werden, was exemplarisch im hellroten Bereich "MITK" dargestellt ist. Quelle: [10]

[2] http://cppmicroservices.org/

Die Hardwareschicht „ToFHardware“ (grün in Abbildung 5) ermöglicht das Ansteuern einer Kamera unterschiedlichster Hersteller. Unterstützt werden momentan unter anderem die Tiefenbildkameras CamCube 3.0[3] und Microsoft Kinect[4]. Die Einbindung weiterer Kameratypen wird mit geringem Aufwand ermöglicht. Alle Geräte sind als C++ Micro Services implementiert, sodass sie Plugin übergreifend genutzt werden können. Außerdem ist die Hardwareschicht für Kamerafunktionalitäten, wie Bildaufnahme und Wiedergabe, und für das Weiterreichen der Rohdaten an die Verarbeitungsschicht zuständig. In der Verarbeitungsschicht „ToFProcessing“ (rot in Abbildung 5) können ToF-Bilddaten durch verschiedenste Filter modifiziert werden. [9] Die Anwendungsschicht ToF-UI (gelb in Abbildung 5) stellt die Schnittstelle zum Benutzer her. Dieses grafische Benutzerinterface (GUI) enthält verschiedene Elemente, die den Zugriff auf die von den unteren Schichten zur Verfügung gestellten Funktionen ermöglichen. Neben der Herstellung einer Verbindung zu einer Tiefenbildkamera bietet diese Benutzeroberfläche die Möglichkeit, Bilddaten aufzunehmen und abzuspielen. Weiterhin können auf diese Daten verschiedene Filter angewendet werden und mehrere Visualisierungsparameter für die Bilddaten eingestellt werden. [10]

2.4 AKQUISITION VON TIEFENDATEN

Der folgende Abschnitt gibt einen Überblick über zwei verschiedene Techniken zur Akquisition von Tiefenbilddaten. Dies ist deshalb für diese Arbeit relevant, da die Verwendung von Tiefenbilddaten einen Grundbaustein der markerlosen mobilen Augmented Reality, deren Visualisierung in dieser Arbeit verbessert werden soll, darstellt. Genauer wird das Konzept der markerlosen mobilen Augmented Reality in Abschnitt 2.5 erläutert.

Die erste Technologie nennt sich strukturiertes Licht. Als strukturiertes Licht versteht man die Projektion eines bekannten Musters auf eine Oberfläche. Anhand der Verzerrung des Musters lassen sich die Krümmung und Tiefeninformationen der Oberfläche berechnen. Diese Technologie findet in der ersten Version der Kamera Kinect aus der Kooperation der Softwarefirma Microsoft und PrimeSense Anwendung. Genauer wird diese Technik in [19] erläutert.

[3] http://www.pmdtec.com/news_media/video/camcube.php
[4] http://www.microsoft.com/en-us/kinectforwindows/

Die zweite Technologie, die hier vorgestellt wird, ist die Time-of-Flight Technologie. Ein Sensor sendet kurzwelliges Infrarotlicht auf eine Szene aus, das von Objekten in der Szene reflektiert wird. Anhand der Dauer zwischen dem Sendezeitpunkt und dem Zeitpunkt der Ankunft des reflektierten Lichtes lässt sich schließlich die Entfernung des Objektes von der Kamera berechnen. Diese Berechnung wird für jedes Pixel des Kamerabildes durchgeführt. Für eine tiefgehendere Beschreibung dieser Technik sei an dieser Stelle auf die Publikation von Robert Lange [20] verwiesen.

2.5 Konzept der mobilen Augmented Reality

Im Allgemeinen steht der Begriff mobile Augmented Reality für die Kombination eines mobilen Endgeräts mit Augmented Reality. Die Motivation eines solchen Ansatzes liegt darin begründet, dass bisherige Augmented-Reality Systeme eher sperrig und daher umständlich in der Verwendung sind. Die Idee basiert auf der Aufnahme einer aktuellen, realen Szene und dem Darüberlegen zuvor aufgenommener Patientendaten aus einer Computertomografie (CT) oder einer Magnetresonanztomografie (MRT)[21]. Um diese Bilddaten relativ zur realen Szene anzuordnen, müssen Komponenten der realen Szene mit dazu korrespondierenden Elementen aus den Bilddaten registriert werden.

Müller et al. stellen in [22] einen markerbasierten Ansatz zur Registrierung der Bilddaten mit der realen Szene vor. Auf dem Patienten werden Marker platziert, die so beschaffen sind, dass sie auf den Bildern eines CT erkennbar sind, um die Markerposition in diesem Datensatz mit der visuell erkennbaren Markerposition der realen Szene registrieren zu können. Verwendete Registrierungsmethoden sind in [22] aufgezeigt.

Im Konzept der mobilen Augmented Reality aus [21] wird auf Marker verzichtet. Stattdessen wird dem mobilen Endgerät eine Tiefenbildkamera vorgeschaltet (vgl. Abbildung 6). Dieses Konzept wird „ToFoscopy“ genannt.

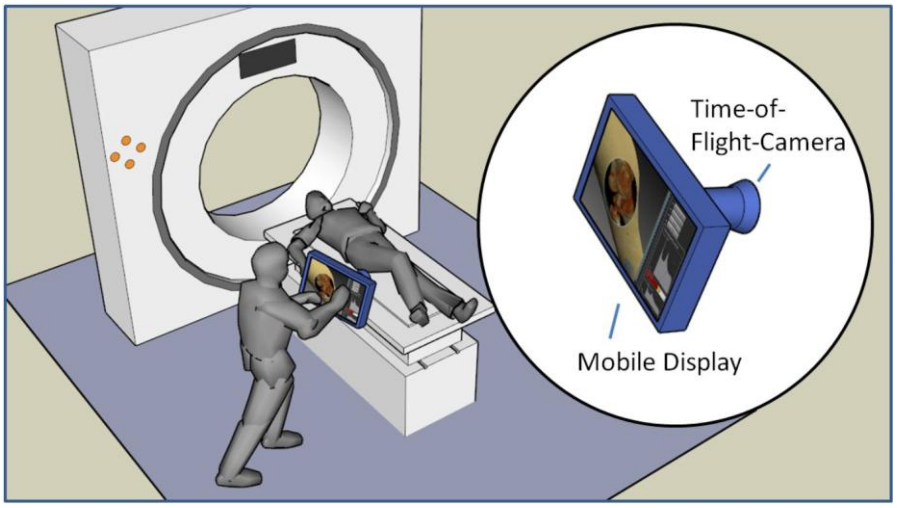

Abbildung 6: Prinzip der ToFoscopy. Einem mobilen Endgerät wird eine Tiefenbildkamera vorgeschaltet. Das Distanzbild aus der Tiefenbildkamera wird zur Registrierung mit der segmentierten Hautoberfläche des Patienten aus zuvor aufgenommenen Bilddaten (CT/MRT) genutzt. Auf dem mobilen Endgerät kann das Farbbild der realen Szene in Kombination mit Augmented Reality Objekten, die ebenfalls aus dem zuvor akquirierten Bilddatensatz gewonnen werden, betrachtet werden. Mithilfe der Registrierung können die virtuellen Objekte relativ zur aktuellen Kameraposition transformiert werden. Quelle: [21]

Die Tiefendaten, die diese Kamera liefert, können zu einer Oberfläche rekonstruiert werden (siehe [23]). Diese Oberfläche wird zur Registrierung mit der Patientenoberfläche, gewonnen aus einem vorher akquirierten Bilddatensatz (z.B. aus dem CT oder MRT), genutzt. Mithilfe dieser Registrierung wird die Patientenanatomie aus den Bilddaten entsprechend der Kameraposition an die aktuelle Lage des Patienten angepasst.

Für die Registrierung der Oberfläche aus den Tiefendaten mit der Patientenoberfläche aus den Bilddaten wurde der Iterative Closest Point (ICP) Algorithmus [24] weiterentwickelt. Der ICP Algorithmus sucht in jeder Iteration neue Korrespondenzen zwischen zwei zu registrierenden Punktwolken $P = \{\overrightarrow{p_1}, \overrightarrow{p_2}, \dots, \overrightarrow{p_N}\}$ und $X = \{\overrightarrow{x_1}, \overrightarrow{x_2}, \dots, \overrightarrow{x_N}\}$. Eine Korrespondenz ist in diesem Zusammenhang der Punkt $\overrightarrow{x_i} \in X$, der der nächste Nachbar zum Punkt $\overrightarrow{p_i} \in P$ ist. Das heißt, dass derjenige Punkt x_i als Korrespondenz ausgewählt wird, dessen euklidische Distanz zu p_i minimal ist. Mit der Methode der kleinsten Quadrate von Horn [25] wird eine Transformation berechnet, die den Fehler der Registrierung der beiden Punktwolken minimiert. Der ICP-Algorithmus terminiert, wenn dieser Fehler unter einen bestimmten Schwellwert fällt.

Die Güte der Registrierung des ICP-Algorithmus ist von der Häufigkeit des Auftretens statistischer Fehler bei der Lokalisierung von Korrespondenzen abhängig. Time-of-Flight Kameras haben beispielsweise eine deutlich größere Unsi-

cherheit bei der Lokalisierung von Korrespondenzen in Blickrichtung der Kamera. Um diese richtungsabhängigen statistischen Fehler bei der Registrierung zu berücksichten, wurde in [26] der anisotrope ICP-Algorithmus entworfen. Dieser berücksichtigt diese statistischen Fehler und zeigte einen deutlich geringeren Registrierungsfehler als der konventionelle ICP Algorithmus. Durch eine Parallelisierung des anisotropen ICP-Algorithmus wurde die Laufzeit des Algorithmus deutlich verkürzt. [27] Für die markerlose, mobile Augmented Reality bedeutet dies, dass eine genaue Feinregistrierung in Echtzeit möglich ist.

3 VERWANDTE ARBEITEN

Der folgende Abschnitt stellt den aktuellen Stand der Technik hinsichtlich Augmented Reality Visualisierungen dar. Um die Tiefenwahrnehmung in einer Szene mit virtuellen Objekten zu verbessern, wurden verschiedene Ansätze entwickelt und evaluiert. Im Folgenden werden drei verschiedene Techniken vorgestellt.

Im ersten Ansatz von Hansen et al. [28] wird stark von der gewohnten Darstellung von Oberflächen abgewichen. Im häufigsten Fall werden Organe als Oberflächen auf die Szene projiziert. In dem hier vorgestellten Ansatz werden Objekte mithilfe ihrer Konturen visualisiert. Abbildung 7 verdeutlicht optisch den Unterschied zwischen konventioneller (links) und neu vorgestellter Darstellung (rechts).

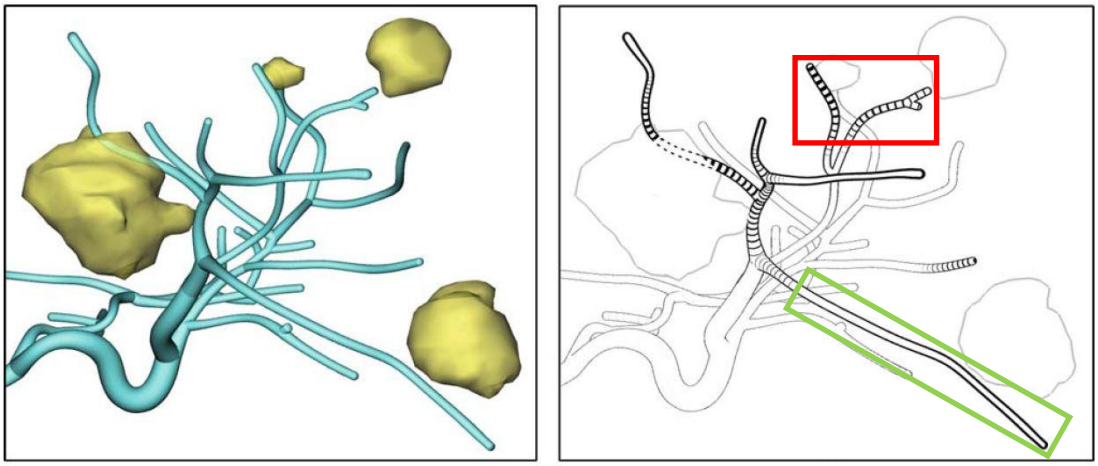

Abbildung 7: Vergleich konventioneller Augmented Reality Visualisierung (links) mit der Visualisierung von Objekten als Kontur (rechts) aus [28]. Verschiedene Strichstärken der Umrisslinien (grüner Rahmen, Strichstärke steigt von links nach rechts) und unterschiedlich stark schraffierte Oberflächen (roter Rahmen, linker Ast wird mit dickeren Linien schraffiert als rechter Ast) sollen eine bessere räumliche Vorstellung der Szene bewirken. Quelle: [28]

Mithilfe unterschiedlicher Strichstärken der Umrisse und Schraffierungen der Oberflächen soll in dieser Idee ein Gefühl von Räumlichkeit in einer Augmented Reality Szene vermittelt werden (vgl. Abbildung 7). Durch eine Evaluierung wurden mehrere Nachteile dieser Darstellung erkannt. Zum einen wirkt eine Szene mit vielen verschiedenen Komponenten (Tumor, Gefäße, Risikostrukturen) durch die verschiedenen Visualisierungstechniken (distanzkodierte Strichstärke und distanzkodierte Schraffierung) schnell überladen. Außerdem wurden bei einer Evaluation dieser Techniken die Parameter der Strichstärke der Um-

risse und der Schraffierung der Oberfläche verwechselt, woraus eine falsche Vorstellung der Gesamtszene resultiert. Dies impliziert, dass dieser stark von der konventionellen Darstellung abweichende Ansatz intuitiv nicht immer korrekt interpretiert wird. Deshalb wurde diese Idee der Augmented Reality Visualisierung in dieser Arbeit nicht in Betracht gezogen.

Lerotic et al. haben in [29] einen Ansatz vorgestellt, in welchem zusätzlich zum eigentlichen virtuellen Objekt dessen reale Umgebung nachgebildet und virtuell dargestellt wird. Die Autoren rekonstruieren die Oberfläche der Umgebung (in Abbildung 8 ist die Umgebung links zu sehen), wobei zur Orientierung des Betrachters beitragende Elemente (beispielsweise Kanten) erhalten werden (Abbildung 8, rechtes Bild).

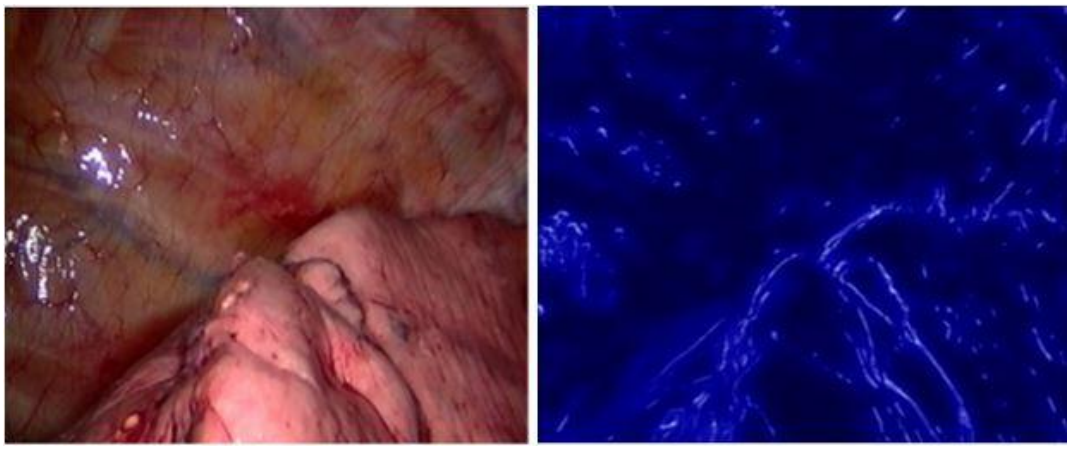

Abbildung 8: Aus der Originalszene (links) wird eine Oberfäche für die Verwendung für Augmented Reality extrahiert und rekonstruiert (rechts). Zur Orientierung des Betrachters beitragende Elemente der Originalszene (hier in hellerem blau dargestellte Kanten) werden beibehalten. Quelle: [29]

Die Kombination von virtuellen Objekten und den Elementen aus der Originalszene trägt dazu bei, dass sich der Betrachter in der Augmented Reality Szene einfacher zurechtfindet (vgl. Abbildung 9).

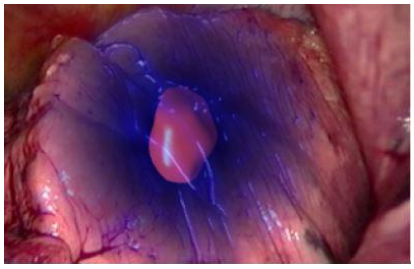

Abbildung 9: Fusion einer virtuellen Oberfläche, die aus der Originalszene rekonstruiert wurde (blau) und virtuellem Objekt (rosa). Eine Erhaltung der Kanten der Originalszene in der virtuellen Oberfläche (blau) erleichtert das Verständnis der Szene. Quelle: [29]

Die Idee der Erhaltung von Merkmalen aus der realen Szene wurde in dieser Thesis aufgegriffen, um dem Betrachter die Orientierung in der Augmented Reality Szene zu erleichtern.

Zuletzt ist noch ein Ansatz von Kalkofen et al. [30] zu erwähnen, bei welchem das Gesamtbild in Untergruppen unterteilt wird. Ziel der Unterteilung ist es, zusammengehörige Objekte zu gruppieren, um sie dann auf dieselbe Art und Weise zu modifizieren. Wie in Kapitel 1.1 erwähnt, führt eine Überlagerung einer Szene mit verdeckten Objekten oft zu einem überladenen Bild, in dem sich der Benutzer kaum noch zurechtfindet. Dieses Problem wird mit diesem Ansatz behandelt. Die Objekte in einer Szene werden in Fokus und Kontext unterteilt. Fokus-Objekte sind solche, auf die der Betrachter sein Augenmerk legen soll, und Kontext-Objekte sind Teile des Bildes, die nicht unbedingt deutlicher gemacht werden sollen, aber trotzdem noch als diese Objekte erkennbar sein sollen, beispielsweise zur Orientierung. Mithilfe dieses Ansatzes kann die Menge an abzubildenden Informationen kontrolliert werden. Durch eine geschickte Kombination der Fokus-und-Kontext-Gruppen kann eine verbesserte Tiefenwahrnehmung erzielt werden. Zur Darstellung von verdeckten Objekten, beispielsweise von inneren Strukturen im Körper, kann die Hautoberfläche als Kontext definiert werden. Für diese Familie kann dann eine bestimmte Darstellungstechnik festgelegt werden, beispielsweise nur Kanten oder Konturen der Oberfläche zu zeigen. Damit bleibt die Struktur der verdeckenden Oberfläche erhalten, wodurch eine Vorstellung der räumlichen Anordnung der verdeckten Strukturen erzielt werden kann. Der Effekt dieser Vorgehensweise ist am Beispiel eines Autos in Abbildung 10 erkennbar [8].

Abbildung 10: Fokus-und-Kontext-Unterteilung am Beispiel eines Autos. Zusammenspiel von Elementen aus der realen Szene (Karosserie, als Kontext definiert) und virtuellen, in der realen Szene unsichtbaren Elementen (Reifen, Motor, als Fokus definiert), um zur Orientierung beitragende Strukturen der Originalszene zu erhalten statt durch virtuelle Objekte zu verdecken. Quelle: [30]

Dieses Konzept legt den Grundstein für den in dieser Arbeit entwickelten Ansatz zur Gruppierung von Objekten, um damit die Menge der abgebildeten Informationen kontrollieren zu können.

4 METHODEN

Das folgende Kapitel beschreibt die Umsetzung der in Kapitel 1.2 beschriebenen Ziele dieser Arbeit.

4.1 ANSATZ

Zunächst wird der Ansatz zur Umsetzung der Problemlösung erläutert.

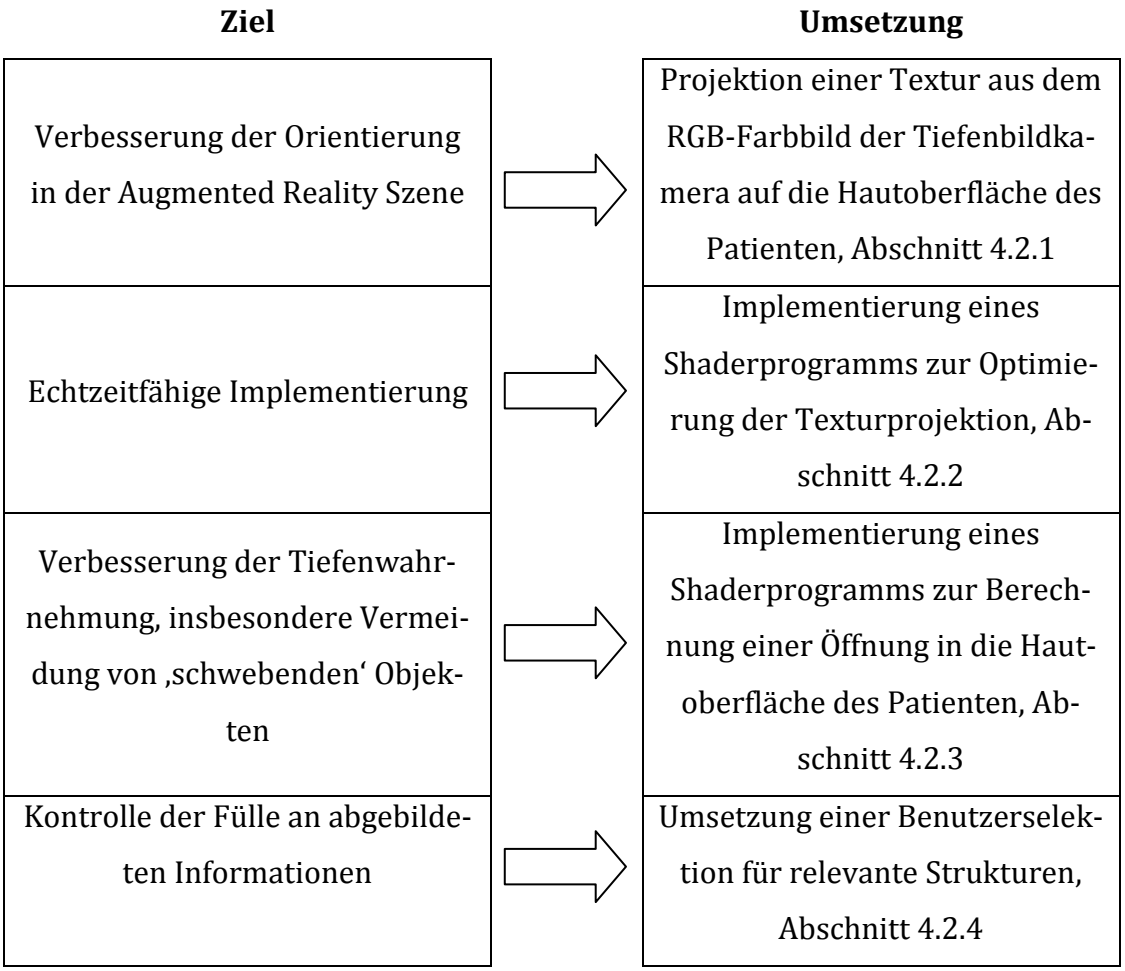

Abbildung 11: Vorgehen zur Umsetzung der definierten Ziele

Um dem Betrachter das Verständnis der Augmented Reality Szene zu erleichtern, wird im ersten Teil der Umsetzung eine Verbesserung der Orientierung in der Augmented Reality Szene angestrebt. Aus einem Farbbild der Szene, welches von einer Tiefenbildkamera aufgenommen wird, wird die daraus gewonnene Farbinformation mithilfe von Texturmapping (vgl. Abschnitt 2.1.1) auf einen virtuellen Körper des Patienten übertragen. Dieser virtuelle Körper stammt aus zuvor akquirierten Bilddaten des Patienten, indem man aus diesen Bilddatensätzen die Haut segmentiert. Das entstandene Objekt kann als wirk-

lichkeitsgetreue Nachbildung der Hautoberfläche des Patienten verstanden werden. Durch eine Projektion dieser Nachbildung in das Farbbild kann die Augmented Reality Szene mit Informationen aus der realen Szene angereichert werden, was zu einer besseren Orientierung in der Szene beiträgt.

Die obige Texturprojektion wird durch ein Shaderprogramm (vgl. Grundlagen, Abschnitt 2.1.2) optimiert. Dieses Programm sorgt dafür, dass je nach Position der Tiefenbildkamera, aus der das Farbbild stammt, genau diejenigen Teile der segmentierten Oberfläche texturiert werden, die entsprechend in der realen Szene von der Tiefenbildkamera erfasst werden können. Die Berechnung der Texturprojektion abhängig von der Lage der Tiefenbildkamera erfordert eine relativ hohe Anzahl an Rechenschritten, weshalb die Berechnung effizient in einem Shaderprogramm umgesetzt wurde.

Weiterhin soll in dieser Arbeit die Tiefenwahrnehmung innerhalb einer Augmented Reality Szene verbessert werden. Hierzu wird eine Öffnung in die Hautoberfläche des Patienten projiziert, die sozusagen einen Blick in das Innere des Patienten ermöglicht und gleichzeitig im Zusammenspiel mit der Patientenoberfläche für eine räumliche Vorstellung der Szene sorgt. Diese Berechnung wird ebenfalls in einem Shaderprogramm realisiert.

Zuletzt wurde ein Konzept umgesetzt, das dazu beiträgt, die Fülle an abgebildeten Objekten in der Szene zu kontrollieren. Hierbei wurde die Idee der Fokus-und-Kontext Unterteilung aus Abschnitt 3 aufgegriffen, wodurch der Benutzer die segmentierten Oberflächen aus zuvor aufgenommenen Bilddaten in verschiedene Gruppen klassifizieren kann. Für jede Gruppe kann eigens entschieden werden, ob die zu dieser Gruppe gehörigen Objekte in der Augmented Reality Szene angezeigt werden. Zusätzlich werden die Gruppen optisch durch unterschiedliche Visualisierungen gegeneinander abgegrenzt, um zu erreichen, dass der Benutzer sich auf eine bestimmte Gruppe von Objekten intuitiv fokussiert und von anderen Objekten so wenig wie möglich abgelenkt wird.

4.2 UMSETZUNG

Wie in Kapitel 1.2 erwähnt, ist der Kernpunkt dieser Arbeit, die Darstellung von Augmented Reality im Kontext der ToFoscopy zu verbessern. Die Logik der ToFoscopy ist in einem eigenen Plugin umgesetzt. Die Augmented Reality Visua-

lisierung wurde in einer extra Klasse, ein Widget namens QmitkToFVisualization, implementiert. Somit wird die Wiederverwendung dieser Implementierung deutlich erleichtert. Ein Widget bedeutet im Kontext von MITK nichts anderes, als die Verbindung einer Implementierung mit einem GUI. Diese Widgets können dann beliebig wiederverwendet werden.

Abbildung 12 veranschaulicht die Einordnung von QmitkToFVisualization in die Architektur von MITK.

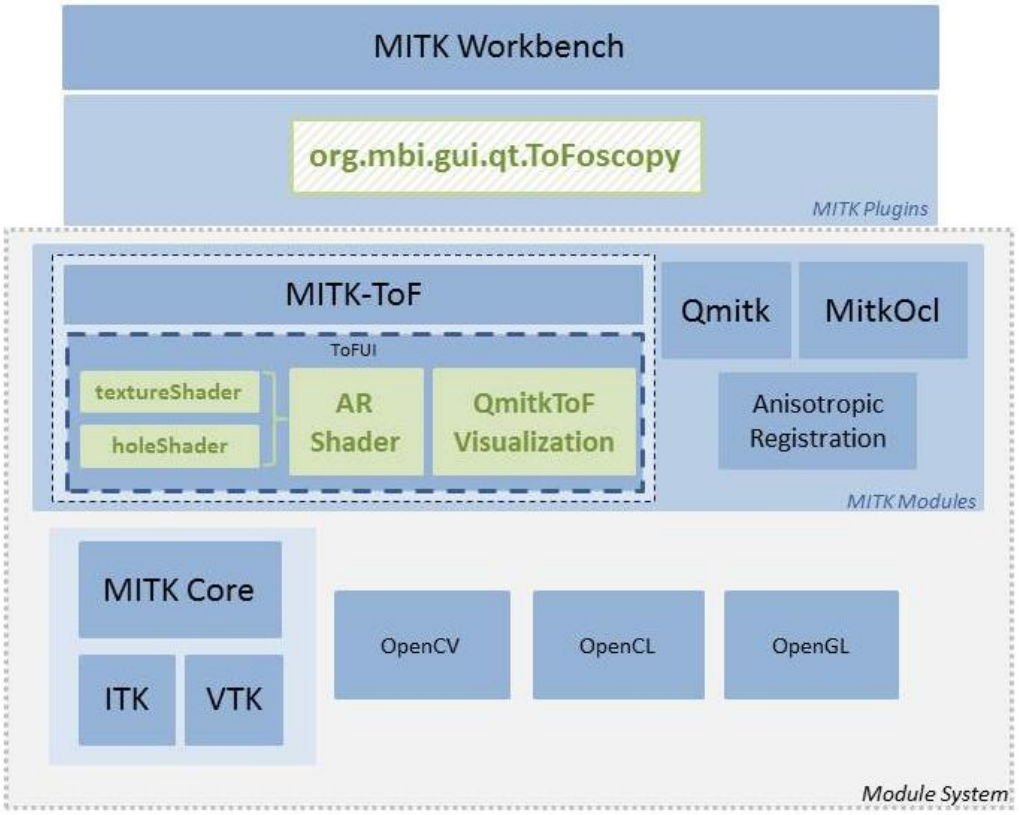

Abbildung 12: QmitkToFVisualization (grün) innerhalb der MITK Architektur. Das in dieser Thesis entwickelte Widget QmitkToFVisualization ist in die Anwendungsschicht von MITK-ToF integriert. Dort befinden sich ebenfalls zwei entwickelte Shaderprogramme (textureShader und holeShader), die sowohl getrennt als auch gemeinsam (AR Shader) nutzbar sind.

Das Plugin ToFoscopy benutzt die Module Qmitk, MITK-ToF, AnisotropicRegistration und MitkOcl. Diese werden für die Benutzung von GUI-Elementen (Qmitk), die Ansteuerung der Hardware und Verarbeitung der Bilddaten (MITK-ToF, vgl. Kapitel 2.3) und die Registrierung der Daten aus der Tiefenbildkamera mit der Oberfläche aus dem zugehörigen Bilddatensatz (AnisotropicRegistration) benötigt. Das Modul MitkOcl [31] stellt die Schnittstelle zwischen MITK und OpenCL her, welches für die Entwicklung paralleler Programme für verschiedene Plattformen benötigt wird. Das im Zuge dieser

Arbeit entwickelte Widget QmitkToFVisualization ist in die Anwendungsschicht ToF-UI des Moduls MITK-ToF (vgl. Kapitel 2.3) integriert. In der Klasse QmitkToFVisualization wurde ein Konzept zur Gruppierung von Augmented Reality Objekten umgesetzt. Weiterhin sind zwei Shaderprogramme (Abbildung 12, textureShader und holeShader) entstanden, die sowohl getrennt als auch kombiniert (Abbildung 12, ARShader) nutzbar sind. Für die Verwendung der Visualisierung im Kontext der ToFoscopy, wurde das entsprechende ToFoscopy-Plugin (in Abbildung 12 grün schraffiert dargestellt) um die nötigen Schritte erweitert. Die MITK Workbench stellt schließlich die Schnittstelle zum Benutzer dar.

4.2.1 TEXTURPROJEKTION

Für die Verbesserung der Orientierung in der Augmented Reality Szene wurde die Projektion einer Textur auf eine Oberfläche umgesetzt. Die Textur stammt aus einem RGB-Farbbild, das generell frei wählbar ist. Im Kontext der ToFoscopy bietet sich aber das von der Kamera generierte Farbbild an. Als Oberfläche, auf die die Textur projiziert wird, ist die Hautoberfläche des Patienten angedacht, die aus zuvor akquirierten Bilddaten des Patienten segmentiert wurde. Die zugrunde liegende Idee ist, dass eine künstliche Nachbildung des Körpers des Patienten geschaffen wird, welcher als Umgebung für weitere virtuelle Objekte, wie beispielsweise Organe des Patienten, dient. Dadurch wird dem Benutzer die Orientierung innerhalb der Augmented Reality Szene erleichtert.

Um eine Textur passend auf ein Objekt zu projizieren, muss für jede dreidimensionale Koordinate eines Punktes auf dem Objekt die dazu korrespondierende Koordinate in der Textur ermittelt werden (vgl. Abschnitt 2.1.1). Diese Berechnung wird mithilfe einer Instanz der VTK-Klasse `vtkProjectedTexture`[5] durchgeführt. Die Verwendung der Klasse erforderte einen hohen Einarbeitungsaufwand, da die Dokumentation dieser Klasse nicht für die korrekte Verwendung ausreichte. Die Funktionsweise wird deshalb im Folgenden detailliert beschrieben.

[5] http://www.vtk.org/doc/nightly/html/classvtkProjectedTexture.html

Abbildung 13 zeigt den für die Berechnung zugrunde liegenden Aufbau schematisch. Das zu erkennende Koordinatensystem repräsentiert dasjenige, welches VTK zur Platzierung und Darstellung von Objekten verwendet. Der blaue Würfel im Bild symbolisiert das dreidimensionale Objekt, für welches Texturkoordinaten berechnet werden sollen. Schwarz im Bild ist die Kamera dargestellt, aus der das zu projizierende Farbbild (rotes Trapez) aufgenommen wird.

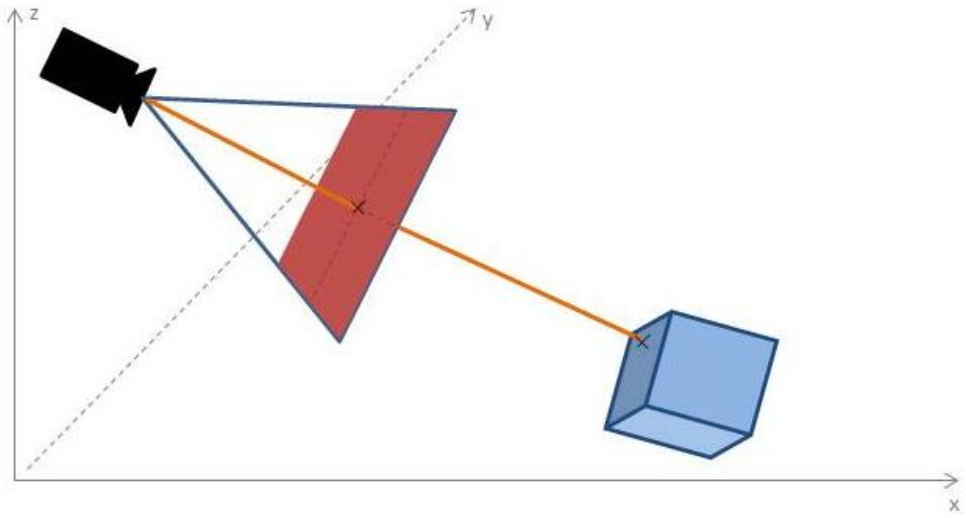

Abbildung 13: Ausgangssituation für die Berechnung von Texturkoordinaten für ein bestimmtes Objekt (blauer Würfel) mithilfe der Klasse `vtkProjectedTexture`. Eine Kamera nimmt ein Farbbild (rotes Trapez) auf. Auf den im Würfel eingezeichneten Punkt soll der Punkt des Farbbildes abgebildet werden, der auf der Geraden zwischen Objektpunkt und Kamera liegt (orange). Das Koordinatensystem wird von VTK zur Beschreibung der Lage aller abgebildeter Objekte verwendet.

Der zu einem Objektpunkt korrespondierende Bildpunkt ist der Schnittpunkt des Farbbildes mit der Geraden, die den Objektpunkt mit der Kamera verbindet (orange Gerade in Abbildung 13). Je nach Position und Orientierung der Kamera schneidet die Gerade das Farbbild in einem anderen Punkt. Dies ist exemplarisch in Abbildung 14 dargestellt.

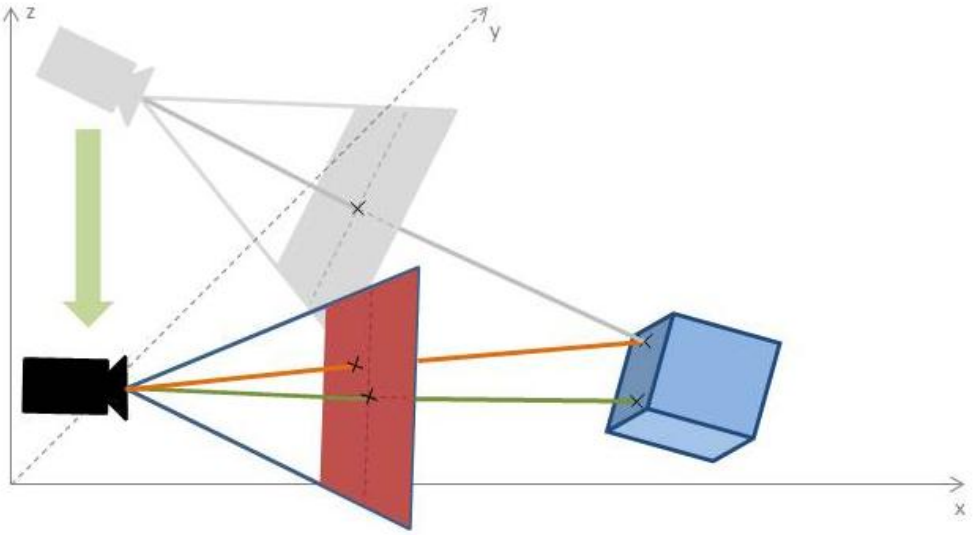

Abbildung 14: Durch die verschobene Kameraposition und –orientierung muss auf den Objektpunkt (hier auf der orangen Gerade) aus Abbildung 13 ein anderer Bildpunkt des Farbbildes gemappt werden.

Mithilfe einer Instanz der Klasse `vtkProjectedTexture` wird nun für jede dreidimensionale Koordinate des Objekts eine zweidimensionale Texturkoordinate berechnet, die den Zusammenhang zwischen Objekt- und Bildpunkten herstellt. Die perspektivisch korrekte Berechnung ist abhängig von der Kameraposition und -orientierung, der Kameraauflösung und der Brennweite der Kamera. Diese Parameter müssen deshalb in der Instanz der `vtkProjectedTexture` gesetzt werden. Zusätzlich benötigt diese Instanz als Eingabe das Objekt, für welches Texturkoordinaten berechnet werden sollen. Es werden außerdem die Intervalle definiert, innerhalb welcher sich die berechneten Texturkoordinaten bewegen. Diese Intervalle nennen sich SRange (in horizontaler Richtung der Textur) und TRange (in vertikaler Richtung der Textur). Häufig werden diese Intervalle auf [0;1] gesetzt, was einer Normierung der Texturkoordinaten entspricht. Hierdurch können die Texturkoordinaten unabhängig von der Auflösung angegeben werden. Die Intervalle sind grafisch in Abbildung 15 dargestellt.

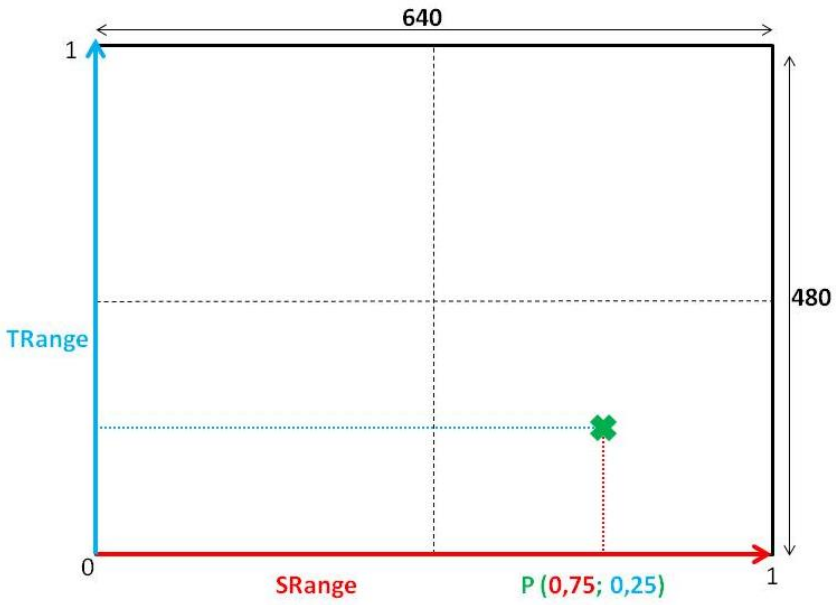

Abbildung 15: Intervalle der Texturkoordinaten in x-Richtung ("SRange" in rot) und y-Richtung ("TRange" in blau) der Textur (exemplarisch als schwarzes Rechteck abgebildet).

Der Vollständigkeit halber ist an dieser Stelle zu erwähnen, dass die obige Definition der Intervalle im Anwendungsfall zu einer "verschobenen" Projektion der Textur auf dem Objekt führt (vgl. Abbildung 16).

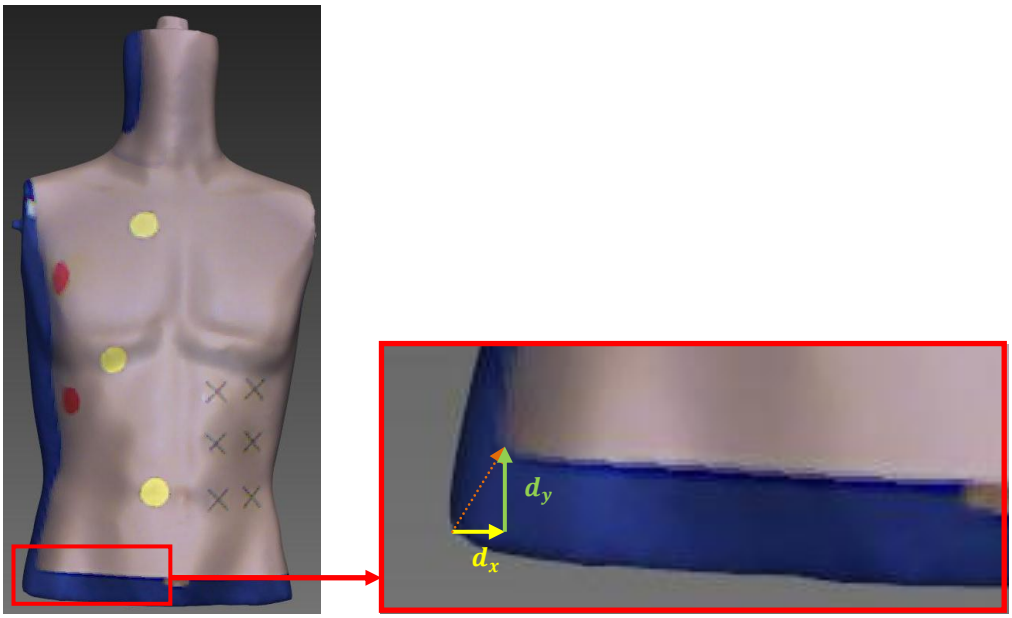

Abbildung 16: Verschobene Texturprojektion, entstanden durch Fertigungstoleranzen der Kamera.

Diese Verschiebung kommt durch Fertigungstoleranzen der verwendeten Kamera für das Farbbild zustande. Im vorliegenden Fall befindet sich der Detektor, der das Bild aufnimmt, etwas verschoben gegenüber der Linse, was eine Verschiebung des Bildmittelpunktes auf dem Detektor bewirkt. Der Mittelpunkt des Detektors liegt nach der Modellvorstellung des Kameraaufbaus auf einer Gera-

den mit der Mitte der Linse. Um diese Verschiebung auszugleichen, musste das Intervall der SRange um den Betrag der Verschiebung in x-Richtung (d_x, gelb in Abbildung 16) verlagert werden. Statt von [0;1] erstreckt sich das Intervall der SRange nun über [0 + d_x;1 + d_x]. Die TRange wird entsprechend um den Betrag der Verschiebung in y-Richtung (d_y) versetzt.

Anhand der gesetzten Parameter berechnet und speichert die Instanz der `vtkProjectedTexture` für jeden Punkt des Objekts die zugehörigen Texturkoordinaten, die dann mit einem Methodenaufruf abgefragt werden können. Für die Zuweisung des Farbbildes, aus dem dann die tatsächliche Farbinformation gewonnen wird, wird in dem zur Oberfläche gehörigen `mitk::DataNode`[6] eine Property gesetzt, die das Farbbild als Eigenschaft dieses DataNodes setzt. Die Kombination von Farbbild und Texturkoordinaten erzeugt eine passgenaue Texturprojektion, wie sie in Abbildung 17 erkennbar ist.

[6] Alle Datenobjekte werden in MITK in einem mitk::DataNode gespeichert. Sie beschreiben die Daten selbst und zugehörige Eigenschaften, sogenannte Properties. Die DataNodes werden im mitk::DataStorage gespeichert, der verschiedene Funktionalitäten zum Suchen, Hinzufügen und Löschen von DataNodes zur Verfügung stellt.

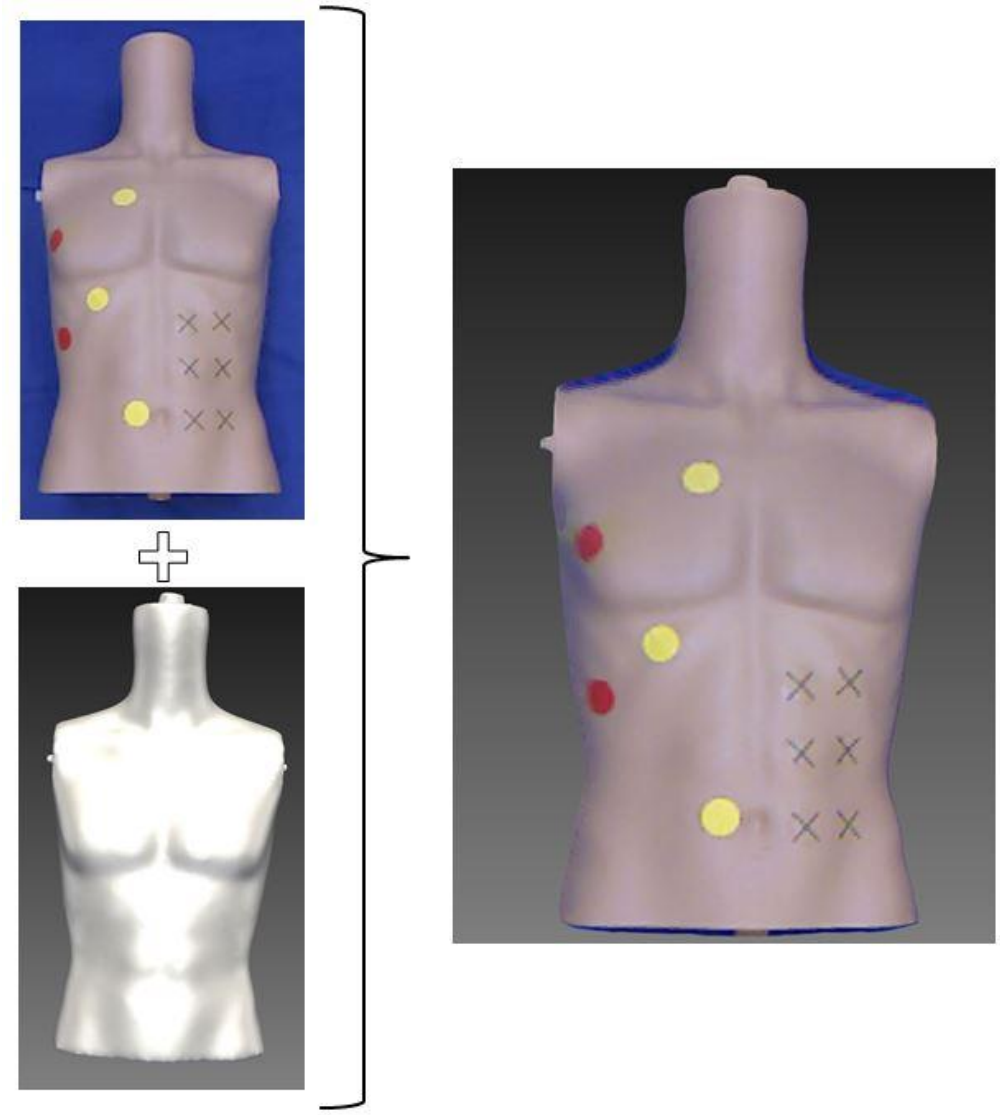

Abbildung 17: Projektion der Information aus einem Farbbild (links oben) auf eine aus zuvor akquirierten Bilddaten segmentierte Patientenoberfläche (links unten). Ergebnis ist eine Fusion der Farbinformation und der Patientenoberfläche (rechts).

Der entstandene virtuelle Körper des Patienten (Abbildung 17, rechtes Bild) kann in das zugehörige Farbbild projiziert werden, um diese Augmented Reality Szene mit Informationen aus der realen Szene zu erweitern. Dies trägt zur Orientierung des Betrachters in der Augmented Reality Szene bei.

4.2.2 Shader zur Optimierung der Texturprojektion

Bei der Berechnung von Texturkoordinaten mithilfe einer Instanz der Klasse `vtkProjectedTexture` aus Abschnitt 4.2.1 ist zu beachten, dass jedem Punkt des Objekts entsprechende Bildpunkte des Farbbildes zugewiesen werden. Abbildung 18 zeigt die Problematik der Berechnung der Texturkoordinaten für ein dreidimensionales Objekt. Dem roten Punkt und dem grünen Punkt wird diesel-

be Texturkoordinate zugewiesen. Das heißt, dass die Textur rundum auf das dreidimensionale Objekt projiziert wird.

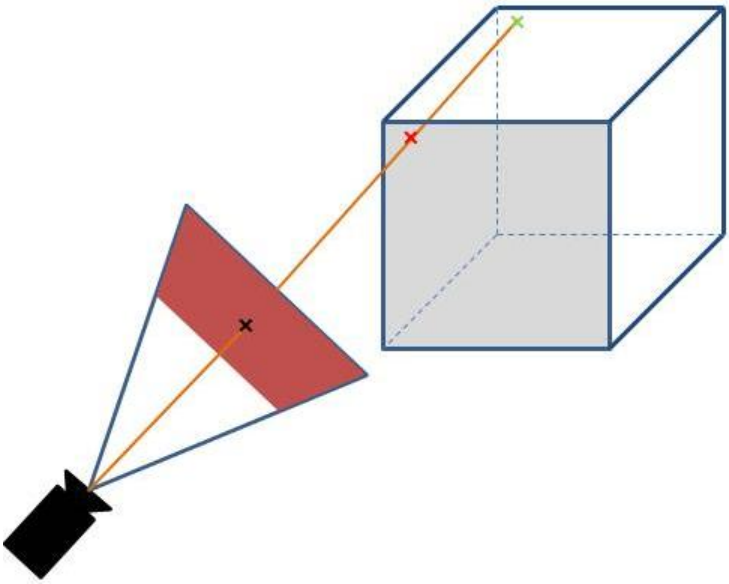

Abbildung 18: Problematik der Texturprojektion mit der `vtkProjectedTexture`. Mehreren Objektpunkten (roter und grüner Punkt) wird dieselbe Texturkoordinate zugewiesen, wenn sie sich von der Tiefenbildkamera gesehen auf einer Geraden (orange in dieser Abbildung) befinden. Resultierend daraus wird die Textur des Farbbildes rundum auf das Objekt projiziert.

In gegebenem Anwendungsfall soll jedoch nur ein bestimmter Teil des Objekts texturiert werden. Das zu texturierende Objekt ist eine aus CT-Daten segmentierte Patientenoberfläche. Je nach Position der Tiefenbildkamera ist jedoch im aufgenommenen Bild nur ein Teil der realen Patientenoberfläche sichtbar. Die Textur sollte dann idealerweise nur auf die Teilfläche der Oberfläche projiziert werden, die der sichtbaren Seite des Patienten entspricht.

Um dieser Problematik entgegenzutreten, wurde ein Shader implementiert, der für jedes Fragment der segmentierten Hautoberfläche des Patienten prüft, ob es einen Teil der realen Patientenoberfläche repräsentiert, der von der Tiefenbildkamera sichtbar ist. Abhängig von dieser Prüfung wird das Fragment texturiert oder nicht.

Das komplette Programm (vgl. Anhang D.1) besteht aus einem Vertex-Shader und einem Fragment-Shader, die kombiniert in einem xml-Dokument gespeichert werden. Die Grundlagen zur Shaderprogrammierung befinden sich in Kapitel 2.1.2.

Im Folgenden wird die Logik des Programms erläutert:

```
11      vertexWorldCoords  = vec3(gl_ModelMatrix * gl_Vertex);
12      normalWorldCoords  = normalize(mat3(gl_ModelMatrix) *
13                                      gl_Normal);
14      vertex             = vec3(gl_ModelViewMatrix * gl_Vertex);
15      normal             = normalize(gl_NormalMatrix * gl_Normal);
16      gl_TexCoord[0]     = gl_MultiTexCoord0;
17      gl_Position        = ftransform();
```

Abbildung 19: Vertex-Shader zur perspektivischen Transformation des Eingangsvertex von Objektkoordinaten zu Projektionskoordinaten.

Der in Abbildung 19 dargestellte Vertex-Shader führt zusammenfassend Transformationen der Eingabewerte in verschiedene Koordinatensysteme durch, und greift auf die zum Vertex gehörige Textur zu. In den Zeilen 11 - 13 werden die Koordinaten des Eingangsvertex und der zugehörigen Normalen in Weltkoordinaten transformiert. In den beiden darauffolgenden Zeilen wird zusätzlich eine Augenpunkttransformation durchgeführt. Diese transformierten Koordinaten werden im Fragment-Shader für die Verarbeitung benötigt, weshalb sie als `varying` Variablen im Vertex-Shader explizit gespeichert werden (vgl. Grundlagen, Abschnitt 2.1.2). Weiterhin wird die Texturkoordinate des Vertex als Eingabeparameter des Fragment-Shaders gesetzt (vgl. Zeile 16). Zuletzt wird mithilfe der Funktion `ftransform()` der Ausgabeparameter `gl_Position` berechnet. Diese Funktion führt eine Transformation von Objektkoordinaten zu Projektionskoordinaten durch (vgl. Grundlagen, Abschnitt 2.1.2).

```
34    if(gl_FrontFacing)
35    {
36       textureColor = texture2D(texSampler,vec2(gl_TexCoord[0]));
37       finalColor = textureColor;
      […]
      }
```

Abbildung 20: Texturierung im Fragment-Shader des Shaderprogramms zur Texturprojektion

Im Fragment-Shader findet die Bearbeitung der Farbinformation eines Fragments statt. Die Abfrage in Zeile 34 bewirkt, dass die darauf folgenden Befehle zur Texturierung nur dann auf das Fragment angewendet werden, wenn es zum Betrachter hin zeigt (Front-Facing). In Zeile 36 wird mithilfe der Funktion

`texture2D` aus der vom Vertex-Shader übergebenen Texturkoordinate die Farbinformation des zu bearbeitenden Fragments berechnet. Die Textur selbst ist in einem sogenannten Sampler (`texSampler` in Abbildung 20) gespeichert.

```
34    if(gl_FrontFacing)
35    {
      […]
48       vec3 d = normalize(projectorPos – vertexWorldCoords);
49       float scalar = d.x*normalWorldCoords.x +
50                d.y*normalWorldCoords.y + d.z*normalWorldCoords.z
51       if (scalar < 0.0)
52          {
53             discard;
54          }
55    }
```

Abbildung 21: Verwerfen nicht sichtbarer Pixel im Fragment-Shader. Mithilfe der Koordinate des Ausgangsvertex wird überprüft, ob die Normale des Ausgangsvertex mit dem Verbindungsvektor zwischen dem Vertex und der Position der Tiefenbildkamera einen Winkel größer 90 Grad einschließen. Wenn ja, wird das zugehörige Fragment verworfen.

Wie bereits erwähnt, ist es bei einem dreidimensionalen Körper wünschenswert, dass die zweidimensionale Textur lediglich auf die Seite projiziert wird, die von der Tiefenbildkamera sichtbar und somit auf dem Farbbild erkennbar ist. Diese Berechnung erfordert mehrere Schritte. Zunächst wird in Zeile 48 (Abbildung 21) der Verbindungsvektor zwischen der Koordinate der Tiefenbildkamera `projectorPos` und der Koordinate des Ausgangsvertex `vertexWorldCoords` berechnet. Hierbei ist zu beachten, dass dieser `vertexWorldCoords` nicht in das Bildschirmkoordinatensystem transformiert wurde, es handelt sich also um die ursprüngliche Position des Vertex in Weltkoordinaten. Es wird das Skalarprodukt aus dem Verbindungsvektor `d` und der Normale `normalWorldCoords` berechnet. Diesem Rechenschritt liegt folgende Überlegung zugrunde: Mathematisch ausgedrückt sind Pixel, die von der Tiefenbildkamera aus sichtbar sind, näherungsweise diejenigen, deren Normalen mit dem Verbindungsvektor `d` zwischen Tiefenbildkamera und Vertex einen Winkel von 90 Grad oder weniger einschließen (vgl. Abbildung 22).

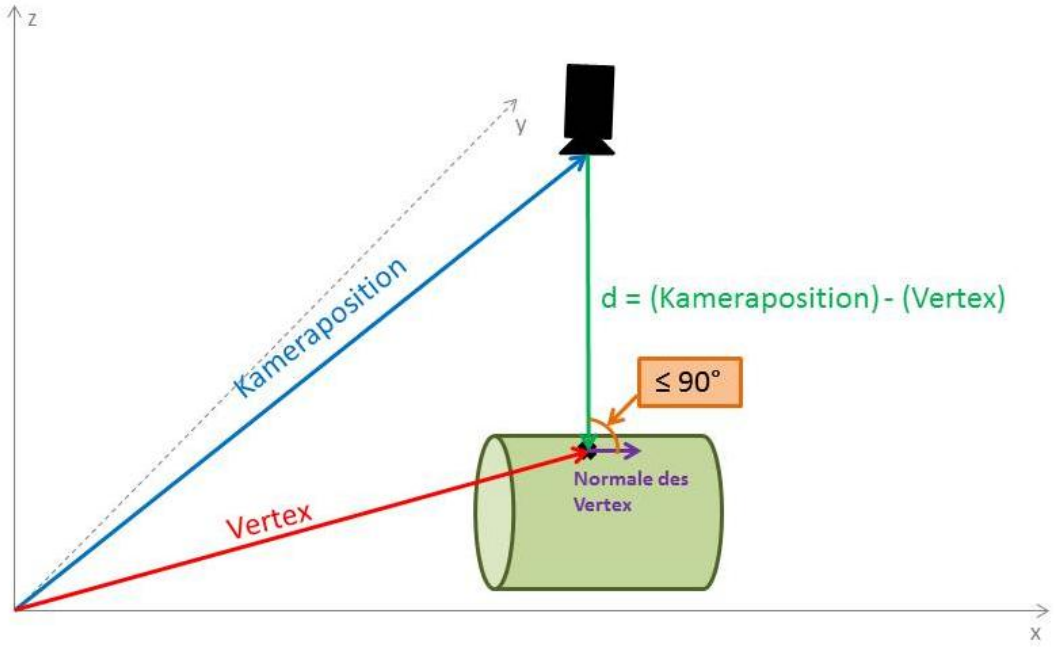

Abbildung 22: Skizze zur Berechnung der Fragmente, die von der Position der Tiefenbildkamera („Kameraposition“, blau) nicht sichtbar sind, und somit verworfen werden sollen. Die zum Vertex gehörige Normale schließt mit dem Verbindungsvektor d (grün) einen Winkel von weniger als 90 Grad ein. Das Fragment gehört somit zu den von der Tiefenbildkamera sichtbaren Fragmenten und soll daher nicht verworfen werden.

In Abbildung 22 ist ein Vertex (schwarzes Kreuz) auf einem Körper abgebildet, der von der Tiefenbildkamera gesehen sichtbar ist. Die Position des Vertex ist durch den roten Vektor „Vertex“ dargestellt. Eine Subtraktion der Position des Vertex von der Kameraposition (Abbildung 22, blauer Vektor) erzeugt den Verbindungsvektor d (grün in Abbildung 22). Nun kann der Winkel zwischen d und dem Normalenvektor des Vertex (lila in Abbildung 22) berechnet werden. In obigem Fall ist dieser Winkel kleiner als 90 Grad. Das Skalarprodukt dieser beiden Vektoren ist dann größer 0. Für Winkel von genau 90 Grad wird das Skalarprodukt 0.

In Abbildung 23 wird der Fall für einen nicht sichtbaren Vertex veranschaulicht. Nun schließen der Verbindungsvektor d und der Normalenvektor des Vertex einen Winkel von mehr als 90 Grad ein. Entsprechend ist das Skalarprodukt dieser beiden Vektoren kleiner 0.

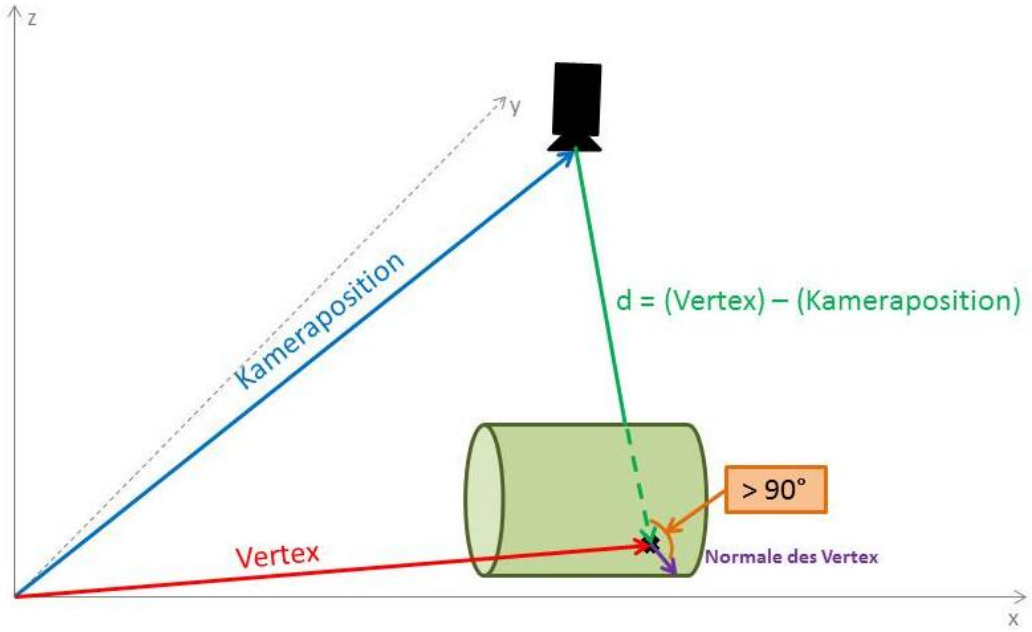

Abbildung 23: Selbige Berechnung wie Abbildung 22 mit einem Vertex, der von der Tiefenbildkamera gesehen nicht sichtbar ist. Der Verbindungsvektor d schließt mit der Normale einen Winkel von mehr als 90 Grad ein. Deshalb soll das Fragment verworfen werden.

Ein Fragment soll genau dann verworfen werden, wenn das Skalarprodukt der zugehörigen Normale mit dem Verbindungsvektor d einen Wert kleiner als 0 hat (vgl. Abbildung 21, Zeilen 51-54). Dieser Effekt ist optisch in Abbildung 25 veranschaulicht.

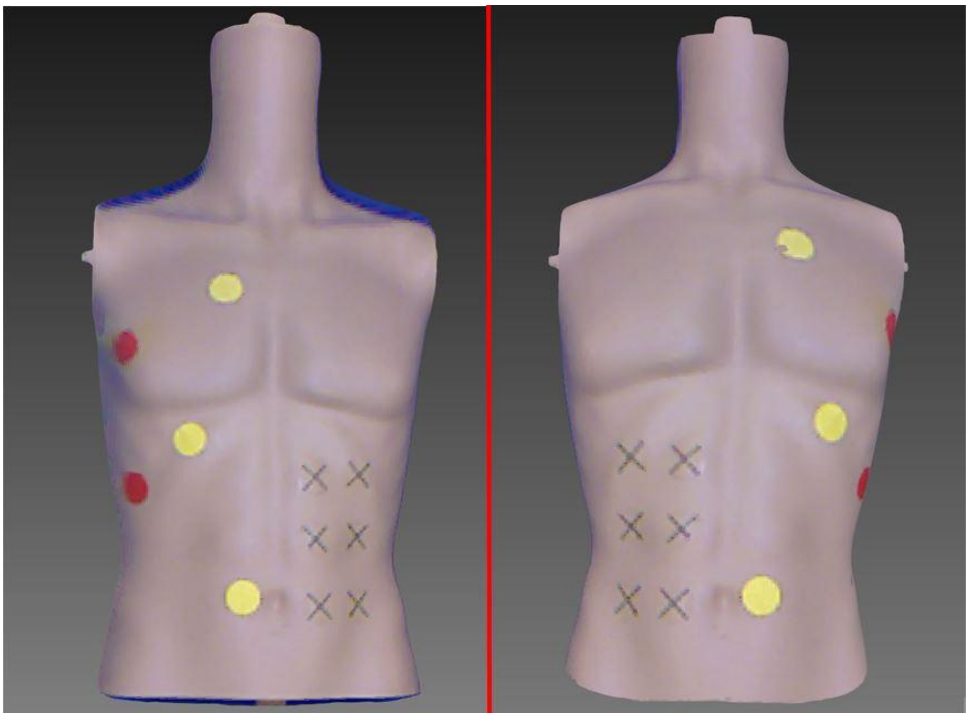

Abbildung 24: Vorher: Farbinformation ist nicht entsprechend der Originalszene nur auf dem Teil der Oberfläche zu sehen, der von der Kamera erfasst werden kann (hier die Vorderseite der Patientenoberfläche (links)), sondern ebenfalls auf der Rückseite der Patientenoberfläche (rechts).

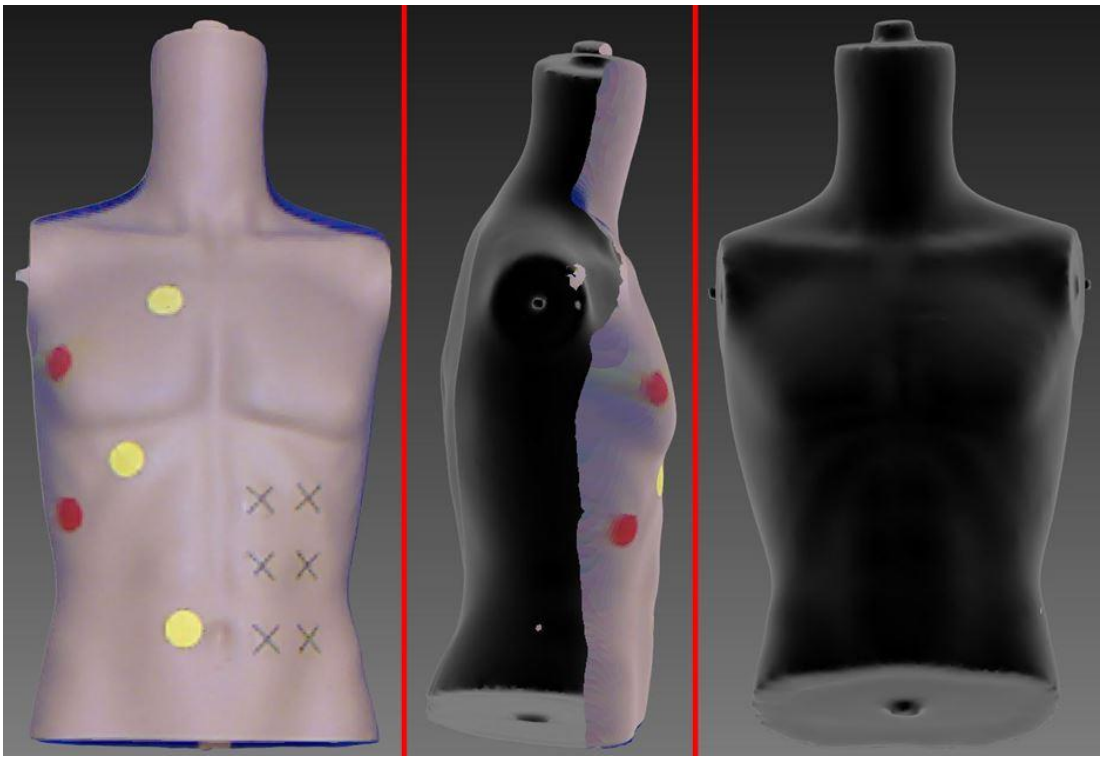

Abbildung 25: Nachher: Optimierte Texturprojektion mithilfe des Shaderprogramms, sodass die Textur des Farbbildes aus der Tiefenbildkamera nur auf die Vorderseite der Patientenoberfläche (links) projiziert wird. Im rechten Bild blickt der Betrachter auf die Rückseite des Körpers. Durch das Verwerfen der Front-Facing Fragmente wird die Rückseite durchsichtig, sodass man nur die Innenfärbung des Körpers erkennt.

Etwas deutlicher kann man den optischen Effekt des Verwerfens von Fragmenten in den roten Rechtecken in Abbildung 26 erkennen. Innerhalb dieser Rechtecke befinden sich Front-Facing Fragmente, die nicht von der Tiefenbildkamera sichtbar sind. Diese Fragmente werden nicht gezeichnet, sodass diese durchsichtig erscheinen. Stattdessen sind nur die dahinterliegenden Fragmente der Innenseite des Körpers zu sehen, also solche, deren Vorderseite nicht zum Betrachter hinzeigt.

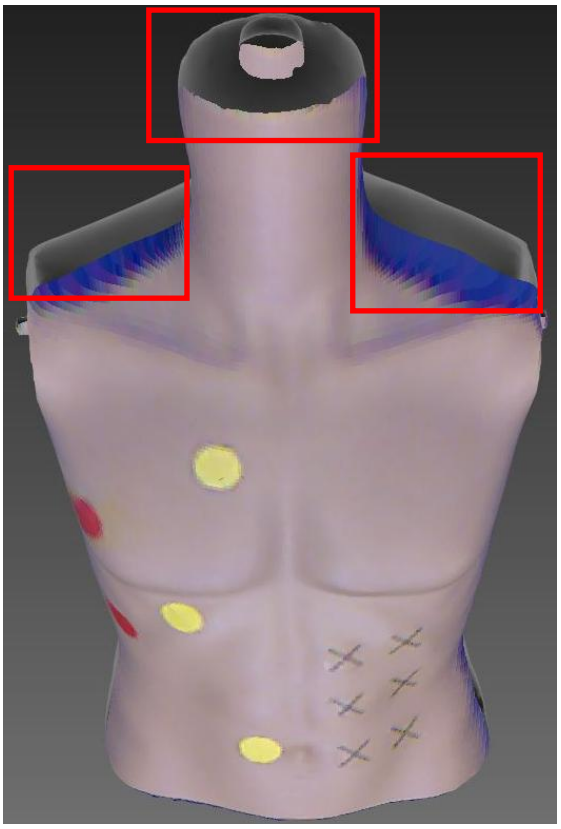

Abbildung 26: Front-Facing Fragmente, die von der Tiefenbildkamera nicht sichtbar sind, werden durchsichtig, sodass nur die dahinterliegenden Fragmente (also die Innenseiten des Körpers) sichtbar sind.

```
33    if(gl_FrontFacing)
34    {
      […]
55    }
56    else
57    {
58       float f = 1-abs(normal.z);
59       f=f/2;
60       finalColor = vec4(f,f,f,1.0);
61    }
62    gl_FragColor = finalColor;
```

Abbildung 27: Färbung des Innenraums des Körpers im Fragment-Shader, um einen guten Kontrast zu den darin befindlichen Organen zu erhalten

Zuletzt wird der Innenraum der Patientenoberfläche eingefärbt. Diese Einfärbung soll für jedes Fragment durchgeführt werden, dessen Vorderseite nicht zum Betrachter zeigt, also nur dann, wenn `gl_FrontFacing` `false` ergibt. Der Ausdruck `normal.z` ist gleichzusetzen mit dem Skalarprodukt des Vektors

`normal` und dem Einheitsvektor z = $\begin{pmatrix}0\\0\\1\end{pmatrix}$ des Kamerakoordinatensystems. Dies entspricht einer Beleuchtung in Kamerarichtung. Das bedeutet, dass

- Flächen, die senkrecht zur Kamera liegen, weiß gefärbt sind
- Flächen, die parallel zur Kamera liegen, schwarz gefärbt sind

Für diesen Anwendungsfall wird der Effekt umgekehrt. Durch die Invertierung `1-abs(normal.z)` sind senkrecht zur Kamera liegende Flächen dunkel, und zur Kamera parallele Flächen hell, sodass der Großteil des Inneren des Körpers dunkel dargestellt wird. Durch die Halbierung von `f` in Zeile 59 wird der Effekt etwas abgeschwächt.

Zum Schluss wird die Farbe in der Variable `finalColor` gesetzt und dann dem Ausgabeparameter `gl_FragColor` übergeben.

Die Innenfärbung des Körpers ist in Abbildung 25 auf Seite 37 im rechten Bild zu erkennen.

Mithilfe des Build-Systems cmake[7] kann der Shader automatisch innerhalb eines Moduls in ein sogenanntes Shaderrepository geladen werden. Hierzu ist lediglich ein Eintrag in die Datei files.cmake nötig. Diese Datei befindet sich im source-Ordner des zugehörigen Moduls und wird um die Pfade der gewünschten Dateien erweitert. In Abbildung 28 ist dies exemplarisch anhand der Dateien "shader1.xml" und "shader2.xml" dargestellt.

```
1   set (RESOURCE_FILES
2     shader1.xml
3     shader2.xml
4     […]
5   )
```

Abbildung 28: Auszug aus der Datei files.cmake zum automatischen Laden des Shaders

Nun muss im Programm, das den Shader aufruft, eine Property in dem Datenobjekt (`node` in Abbildung 29) gesetzt werden, auf welches der Shader angewen-

[7] http://www.cmake.org/

det werden soll (vgl. Abbildung 29, Zeile 1). Über denselben Weg können die uniform Attribute (vgl. Abschnitt 2.1.2) des Shaders gesetzt werden (Zeile 2).

```
1    node->SetProperty("shader",
                       mitk::ShaderProperty::New("textureShader"));
2    node->SetProperty("shader.textureShader.HoleSize",
                       mitk::FloatProperty::New(90.0f));
```

Abbildung 29: Zugriff auf den Shader innerhalb des Programms, das den Shader aufruft

4.2.3 Shader zur Berechnung einer Öffnung in der Hautoberfläche

Um der Augmented Reality Szene eine verbesserte Tiefenwahrnehmung zu geben, wird mittels eines Shaderprogramms eine Öffnung in die virtuelle Hautoberfläche des Patienten gezeichnet. Dies gewährt dem Betrachter eine Art Röntgenblick in das Innere des Patienten.

Das komplette Programm befindet sich im Anhang D.2 und wird im Folgenden erläutert.

```
18      vertex              = vec3(gl_ModelViewMatrix * gl_Vertex);
19      normal              = normalize(gl_NormalMatrix * gl_Normal);
20      gl_Position         = ftransform();
```

Abbildung 30: Vertex-Shader zur perspektivischen Transformation des Eingangsvertex von Objektkoordinaten zu Projektionskoordinaten.

Der Vertex-Shader aus dem Programm holeShader.xml unterscheidet sich nur wenig von dem des Programms textureShader.xml. Für eine Beschreibung der implementierten Logik sei auf Abschnitt 4.2.2 verwiesen.

```
32 if(gl_FrontFacing)
33 {
34    float distanceZAxis = sqrt(dot(vertex.xy,
35                               vertex.xy));
36    if(distanceZAxis < HoleSize)
37    {
38       discard;
39    }
40    if (abs(HoleSize – distanceZAxis) < 2.5)
41    {
42       finalColor = vec4(0.97,0.83,0.35,1.0);
43    }
   […]
   }
```

Abbildung 31: Fragment-Shader: Berechnung einer Öffnung in die Hautoberfläche des Patienten um eine verbesserte Tiefenwahrnehmung zu erreichen

Im Fragment-Shader wird der Oberfläche, auf welche das Shaderprogramm angewendet wird, eine kreisförmige Öffnung hinzugefügt, durch welche der Betrachter in das Innere des Patienten blicken kann (vgl. Abbildung 32).

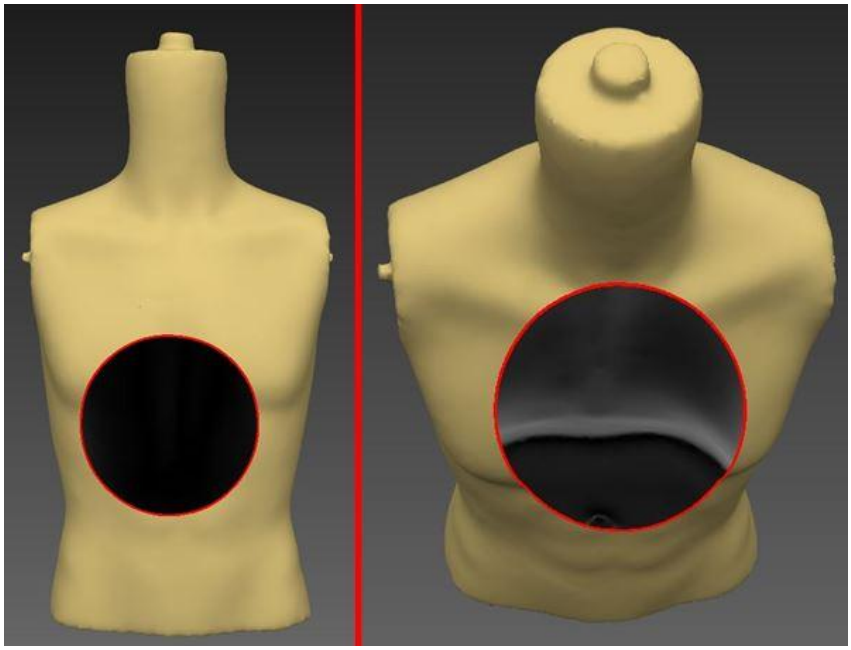

Abbildung 32: Projektion einer Öffnung in eine aus Bilddaten segmentierte Oberfläche mithilfe eines Shaderprogramms, um eine Tiefenwahrnehmung in der Szene zu vermitteln. Die Öffnung verschiebt sich je nach Lage der Oberfläche.

Die Größe der Öffnung kann vom Benutzer gewählt werden, indem er den gewünschten Radius angibt. Innerhalb des Shaderprogramms wird der Radius im Attribut `HoleSize` gespeichert. Der Mittelpunkt der Öffnung befindet sich im Ursprung des Weltkoordinatensystems von OpenGL. Für jedes Fragment, dessen Vorderseite zum Betrachter hin zeigt (vgl. Abbildung 31, Zeile 32), wird in Zeile 36 abgefragt, ob es sich innerhalb des für die Öffnung vorgesehenen Bereiches befindet. Dies ist genau dann der Fall, wenn der Abstand des `vertex` vom Mittelpunkt der Öffnung kleiner ist als der gewünschte Radius. Um zu erreichen, dass diese Öffnung kreis- und nicht kugelförmig gezeichnet wird, wird die zweidimensionale Koordinate (x,y) des `vertex` für die Berechnung herangezogen. In der Variable `distanceZAxis` wird der Abstand des Vertex von der z-Achse gespeichert (vgl. Zeile 34). Dieser Rechenschritt lässt sich etwas verständlicher wie folgt schreiben:

Gegeben sei der dreidimensionale Punkt P mit den Koordinaten $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$. Der Abstand d zur z-Achse lässt sich dann ausdrücken durch:

$$d = \sqrt{x^2 + y^2}$$

Wenn `distanceZAxis` kleiner ist als der Radius der Öffnung, wird das Fragment mit dem Befehl `discard` verworfen und somit nicht ins Ergebnisbild gezeichnet. In den Zeilen 40-43 (Abbildung 31) wird ein farbiger Rand um die Öffnung gezeichnet, um diese optisch deutlicher von der restlichen Oberfläche abzuheben.

Alle Fragmente, die nicht zum Betrachter hinzeigen (also sozusagen die Innenseiten der Oberfläche), werden wie im textureShader aus Abschnitt 4.2.2 eingefärbt (vgl. Abbildung 27 aus Abschnitt 4.2.2).

4.2.4 Benutzerselektion relevanter Strukturen

Um auf die Fülle der abgebildeten Augmented Reality Objekte Einfluss nehmen zu können, wurde ein Konzept umgesetzt, das an das Fokus-und-Kontext-Konzept aus Abschnitt 3 anlehnt. Der Benutzer kann aus zuvor akquirierten Bilddaten segmentierte Objekte verschiedenen Gruppen zuordnen. Zunächst wird der Fokus ausgewählt, also das oder die Objekte, auf welche das Hauptau-

genmerk gerichtet ist. Weiterhin können Objekte als lokaler Kontext und globaler Kontext unterteilt werden. Zum lokalen Kontext gehören Objekte, die sich nah an der Fokusstruktur befinden. Für den globalen Kontext sind solche Objekte vorgesehen, die weiter entfernt von der Fokusstruktur liegen. Diese Gruppen werden jeweils unterschiedlich visualisiert, um die verschiedenen Familien optisch gegeneinander abzugrenzen. Dadurch wird dem Betrachter die komplette Szene schnell verständlich. Weiterhin kann der Benutzer entscheiden, welche der Gruppen tatsächlich angezeigt werden. Es können also Kombinationen gewählt werden, beispielsweise ob nur die Fokusobjekte eingeblendet werden, oder ob der globale Kontext zur Orientierung angezeigt werden soll, und zusätzlich der lokale Kontext, um Risikostrukturen erkennen zu können. Je nach Anwendungsfall wird somit die benötigte Information bereitgestellt, wobei die Anzahl der abgebildeten Objekte berücksichtigt werden kann. So ist es möglich, die Fülle der angezeigten Informationen zu kontrollieren und somit zu vermeiden, dass das Bild überladen wirkt. Durch eine optische Abgrenzung der Fokus-Objekte von den Kontext-Objekten wird zudem gewährleistet, dass der Betrachter nicht von anderen Objekten von seinem Hauptaugenmerk abgelenkt wird.

Alle veränderten DataNodes werden einem StandAlone DataStorage hinzugefügt. Dieser wird der Rendering Pipeline zur Anzeige der Augmented Reality Objekte übergeben, was bereits in einem bestehenden Modul implementiert ist. Die Transformation der Objekte relativ zu den Daten der Tiefenbildkamera ist bereits im Plugin ToFoscopy umgesetzt, sodass hier keine weitere Berechnung der Transformation nötig ist. Es ist ausreichend, diese Transformation auf jedes Objekt anzuwenden, wofür MITK bereits entsprechende Funktionen anbietet.

5 ERGEBNISSE

Das folgende Kapitel gibt Aufschluss über die in dieser Arbeit erzielten Ergebnisse.

5.1 FOKUS-UND-KONTEXT-KONZEPT

Der folgende Abschnitt veranschautlich die Ergebnisse hinsichtlich des in Abschnitt 4.2.4 beschriebenen Fokus-und-Kontext-Konzepts.

Über das in Abbildung 33 abgebildete GUI kann der Benutzer Objekte in drei Gruppen einordnen. Als Fokusstruktur ist ein Lebertumor ausgewählt. Die Strukturen, die sich in unmittelbarer Nähe befinden, hier also der Gefäßbaum der Leber, die Gallenblase, die Leber selbst und der Magen, sind in die Kategorie des lokalen Kontextes eingeordnet. Als weiter entferne Objekte werden hier die Bauchaorta, die Milz, die Nieren und die knöchernen Strukturen des Torso definiert.

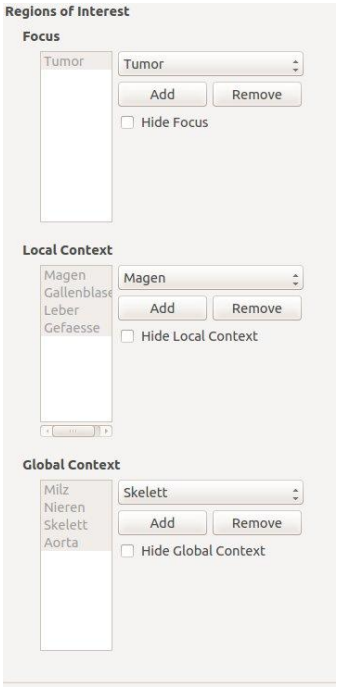

Abbildung 33: GUI zur Gruppierung der in der Augmented Reality Szene dargestellten Objekte. Fokusstrukturen sind solche, auf denen das Hauptaugenmerk liegt. Als Local Context sind umliegende Objekte der Fokustruktur definiert und Global Context Objekte sind solche, die weiter entfernt von der Fokusstruktur liegen.

Abbildung 35 zeigt die unterschiedliche Visualisierung der verschiedenen Familien in der Augmented Reality Szene. Die Fokusstruktur (grün) hebt sich optisch vom lokalen Kontext ab (in Rottönen). Zur Einordnung in den Gesamtkontext

der Szene dient die Anzeige des globalen Kontext. Im Vergleich zur bisherigen Darstellung von Augmented Reality ohne Gruppierung (vgl. Abbildung 34), wird deutlich, dass mithilfe der Einteilung in verschiedene Familien das Hauptaugenmerk des Benutzers nun auf eine bestimmte Struktur gelenkt werden kann.

Abbildung 34: Vorher: Bisherige Augmented Reality Visualisierung ohne Gruppierung. Hier ist nicht erkennbar, welche Strukturen für den Betrachter und den Anwendungsfall relevant sind.

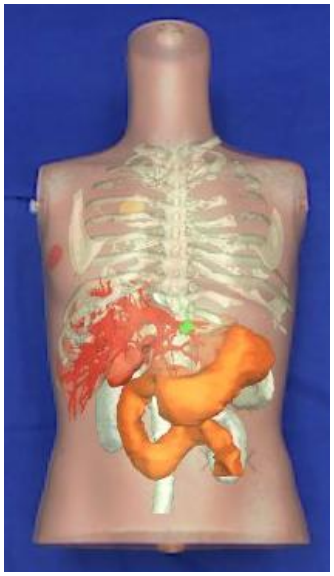

Abbildung 35: Nachher: Die optische Unterteilung zwischen Fokus (grün), lokalem Kontext (rot) und globalem Kontext (weiß) hilft, das Hauptaugenmerk des Betrachters auf eine bestimmte Struktur zu lenken, sodass dieser relevante Strukturen schnell erkennt und von anderen umliegenden Strukturen nicht abgelenkt wird.

Zusätzlich kann der Benutzer über die im GUI erkennbaren Auswahlkästchen entscheiden, welche Gruppen tatsächlich visualisiert werden sollen. Per Knopfdruck kann hierdurch eine Familie komplett aus dem Bild genommen werden

(vgl. Abbildung 36), sodass die ganze Szene weniger befüllt ist (vgl. Abbildung 37).

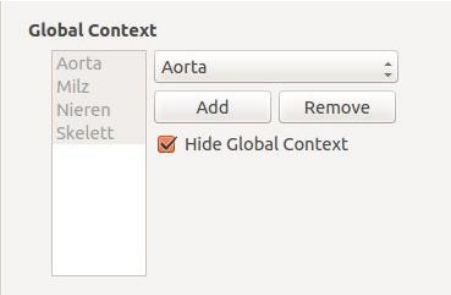

Abbildung 36: Auswahlmöglichkeit über das GUI, Gruppen zu verbergen

Abbildung 37: Visualisierung von Fokus (grün) und lokalem Kontext (rot) ohne globalen Kontext. Das Bild wirkt hierdurch weniger überladen und eine Beschränkung auf die wesentlichen Strukturen wird ermöglicht.

Insgesamt wird dem Benutzer eine Möglichkeit eröffnet, individuell zu entscheiden, welche Objekte in der virtuellen Szene abgebildet werden. Durch die Einordnung in Gruppen kann für eine zusammengehörige Familie von Objekten entschieden werden, ob deren Darstellung erwünscht ist. Hierdurch kann die Fülle an abgebildeten Informationen kontrolliert werden, sodass zum einen ein überladenes Bild vermieden und gleichzeitig verhindert wird, dass der Betrachter von zu vielen abgebildeten Objekten abgelenkt ist. Je nach Anwendungsfall kann entschieden werden, welche Strukturen relevant sind und mit einer entsprechenden optischen Darstellung in die Augmented Reality Szene eingefügt werden sollen.

5.2 TEXTURPROJEKTION

Der folgende Abschnitt demonstriert die Ergebnisse der Texturprojektion aus den Kapiteln 4.2.1 und 4.2.2. Aus der Farbinformation des Farbbildes (generiert von der Tiefenbildkamera) und der segmentierten Patientenoberfläche aus zuvor aqkuirierten Bilddaten ist eine Nachbildung des realen Körpers in Form eines künstlichen Körpers entstanden. Diese Nachbildung wird in das Farbbild projiziert und passt sich durch eine Registrierung (vgl. Abschnitt 2.5) passgenau an das Farbbild an (vgl. Abbildung 39, unteres Bild). Hierdurch wird dem Betrachter die Orientierung in der Augmented Reality Szene erleichtert. Dies lässt sich besonders gut im Vergleich mit der vorherigen Visualisierung der Patientenoberfläche im Farbbild (vgl. Abbildung 38) zeigen. Die Patientenoberfläche in Abbildung 38 überlagert das Farbbild und verdeckt somit Informationen aus dem Bild. In Abbildung 39 ist hingegen zu erkennen, dass Informationen aus der realen Szene aufgegriffen wurden, und auf den virtuellen Körper übertragen wurden. Dadurch findet sich der Betrachter besser in der Augmented Reality Szene zurechtt.

Abbildung 38: Vorher: Bisherige Visualisierung der Patientenoberfläche im Farbbild. Das virtuelle Objekt verdeckt Informationen aus der realen Szene, sodass Informationen daraus verloren gehen.

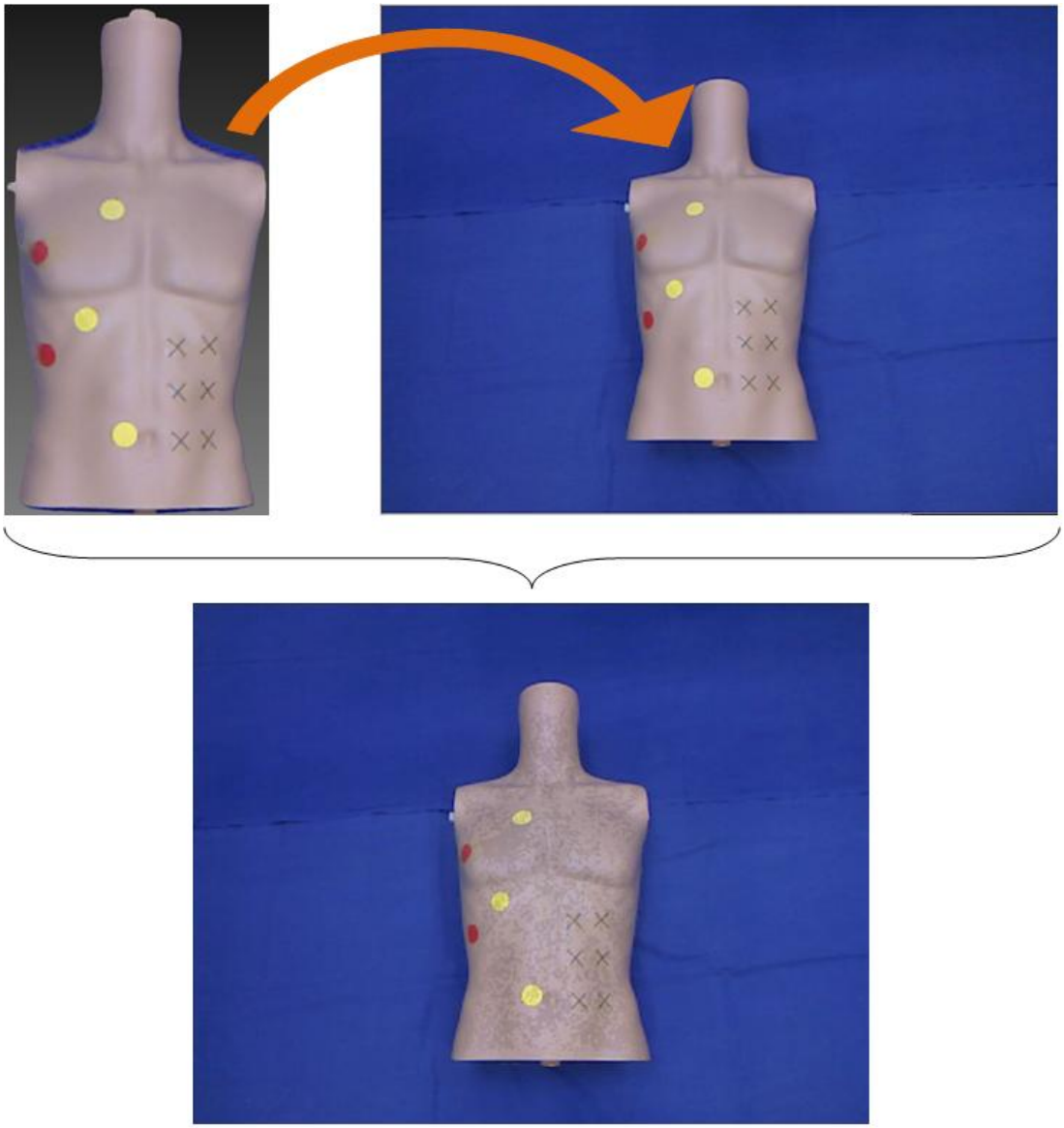

Abbildung 39: Nachher: Unten: Einblendung des virtuellen, texturierten Körpers (oben links) in das Farbbild (oben rechts). Statt das Farbbild zu überlagern, werden Informationen der realen Szene aufgegriffen und mit dem virtuellen Körper so kombiniert, dass dieser dem Körper in der realen Szene entspricht.

Da sich die texturierte Oberfläche nahtlos in das Farbbild einfügt, wurde sie in Abbildung 39 so modifiziert, dass sie im direkten Vergleich mit dem Farbbild erkennbar ist. Dies äußert sich durch die sichtbar dunkleren Stellen im unteren Bild aus Abbildung 39. Etwas besser ist dieser virtuelle Körper in einem Video zu erkennen, das sich auf der angefügten DVD befindet. Dieses Video beinhaltet eine Demonstration der markerlosen mobilen Augmented Reality mit der in dieser Arbeit entwickelten Visualisierung.

5.3 Projektion einer Öffnung in die Hautoberfläche des Patienten

Abbildung 40 zeigt die Projektion einer Öffnung auf die Haut des Patienten. Die räumliche Anordnung der inneren Organe ist durch die Öffnung in Kombination mit der Oberfläche gut zu erkennen.

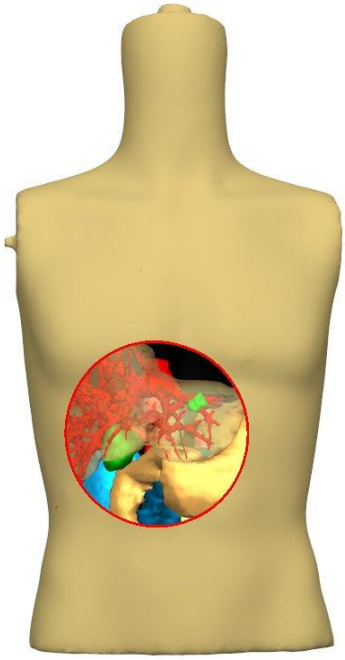

Abbildung 40: Projektion einer Öffnung auf die aus zuvor aufgenommenen Bilddaten segmentierte Hautoberfläche des Patienten zur Verbesserung der Tiefenwahrnehmung.

Wie in Abbildung 41 zu erkennen, ist der Radius der Öffnung flexibel einstellbar. Außerdem ist dort sichtbar, dass sich die Position der Öffnung relativ zur Lage des Patienten verändert.

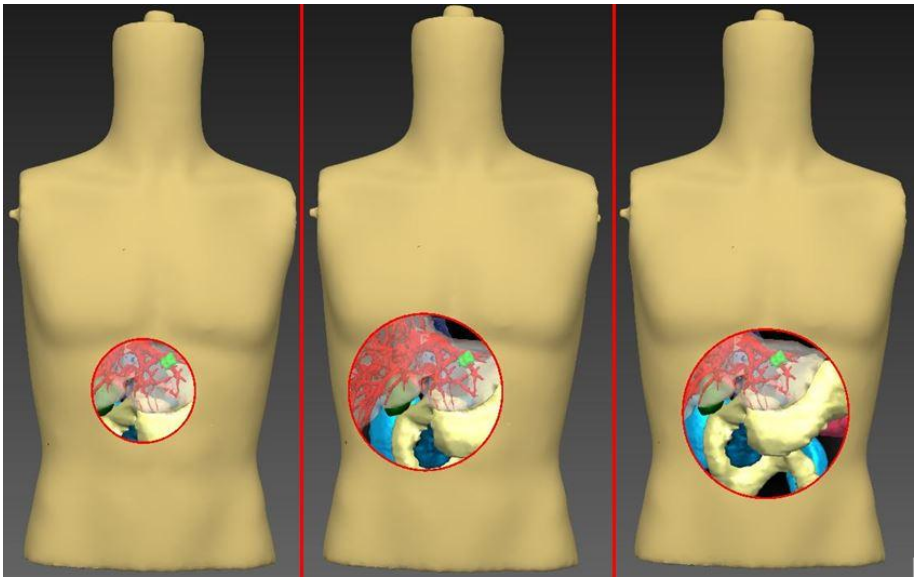

Abbildung 41: Berechnung der Öffnung in die CT-Hautoberfläche des Patienten, um die Tiefenwahrnehmung in der Augmented Reality Szene zu verbessern. Der Radius ist flexibel einstellbar. Die Position der Öffnung passt sich der Blickrichtung des Betrachters an.

Verglichen mit der bisherigen Darstellung in der mobilen Augmented Reality (vgl. Abbildung 42) ist zu erkennen, dass die Objekte nun nicht mehr oberhalb der Szene platziert erscheinen. Mittels der Öffnung gewinnt das Bild an Tiefenwahrnehmung.

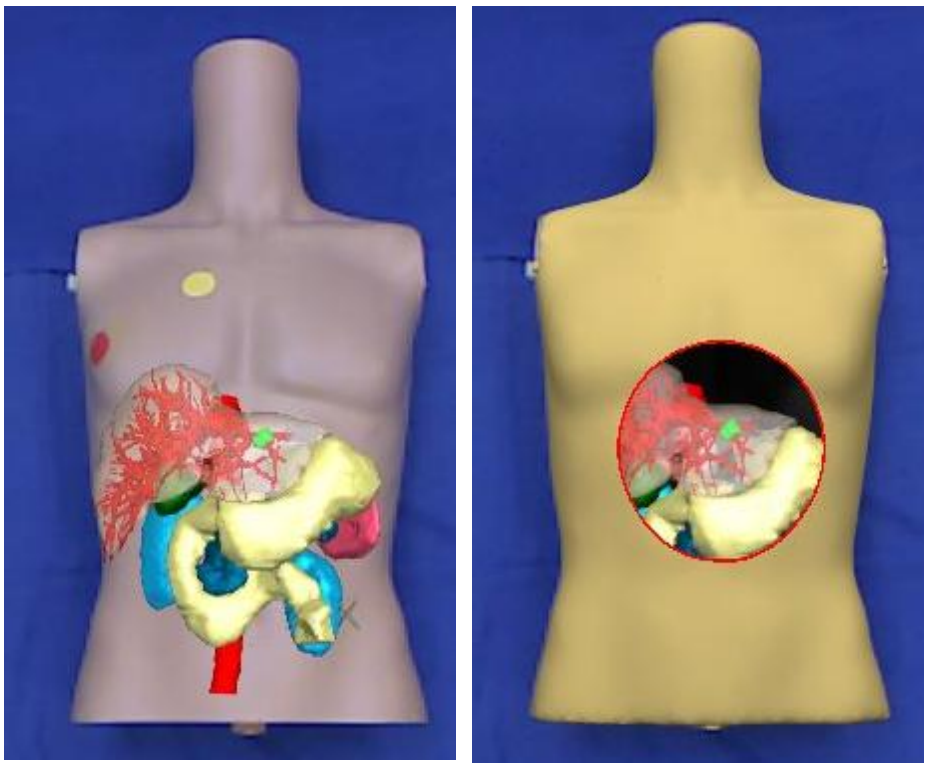

Abbildung 42: Links: Vorher: Augmented Reality mit geringer Tiefenwahrnehmung, sodass der Eindruck entsteht, dass die Augmented Reality Objekte über der Haut schweben. Rechts: Nachher: Verbesserte Tiefenwahrnehmung in der Augmented Reality Szene durch die Kombination aus einer Öffnung und einer Oberfläche. Diese Kombination ermöglicht die räumlich korrekte Einordnung der Organe.

5.4 Kombination der Konzepte

Abbildung 44 zeigt eine Kombination der vorgestellten Konzepte angewendet in der markerlosen mobilen Augmented Reality. Die Textur aus einem Farbbild wird auf die Hautoberfläche projiziert, wodurch eine virtuelle Nachbildung des Körpers des Patienten entsteht. Durch die Projektion dieser Nachbildung in das Farbbild wird eine bessere Orientierung in der Augmented Reality Szene (also dem Farbbild angereichert mit virtuellen Objekten) erreicht. Mithilfe einer Öffnung kann eine Art Röntgenblick in den Patienten gewährleistet werden, sodass die inneren Organe des Patienten betrachtet und in Relation zu dem virtuellen Körper gesetzt werden können. Dies erleichtert die räumlich korrekte Einordnung der Organe. Insbesondere der Vergleich zur vorherigen Darstellung in der mobilen Augmented Reality (vgl. Abbildung 43) zeigt, dass mithilfe der in dieser

Arbeit entwickelten Konzepte die Tiefenwahrnehmung in der virtuellen Szene verbessert werden konnte.

Abbildung 43: Vorher: Mobile Augmented Reality mit geringer Tiefenwahrnehmung, sodass die virutellen Objekte als oberhalb der Szene platziert wirken.

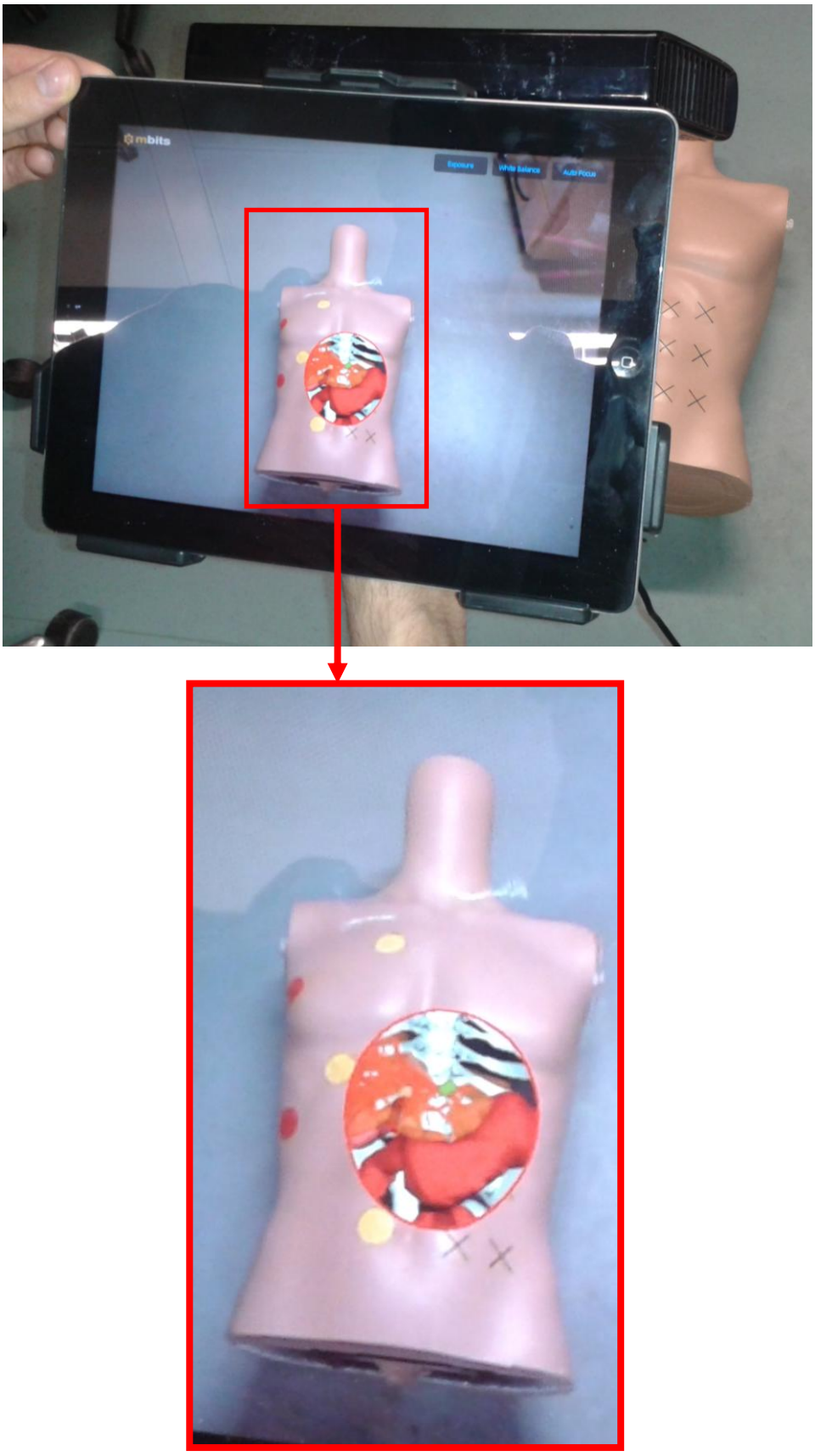

Abbildung 44: Nachher: Mobile Augmented Reality mit verbesserter Tiefenwahrnehmung. Durch eine projizierte Textur auf die Hautoberfläche des Patienten wird die Orientierung in der Augmented Reality Szene erleichtert. Eine Öffnung in die Hautoberfläche ermöglicht eine korrekte räumliche Einordnung der Organe. Die Anwendung des Fokus-und-Kontext Konzepts führt zu einer optischen Unterteilung der Organe in verschiedene Gruppen.

6 DISKUSSION & AUSBLICK

Das Ziel dieser Arbeit war die Umsetzung einer Augmented Reality Visualisierung, die bestimmte Probleme bereits vorhandener Darstellungen vermindert und zugleich echtzeitfähig ist, damit eine effiziente Nutzung im Kontext der mobilen Augmented Reality möglich ist. Zu den Problemen bisheriger Darstellungen zählt zum einen eine fehlende Tiefenwahrnehmung der virtuellen Objekte in der echten Szene und zum anderen eine fehlende Möglichkeit, die Fülle an abgebildeten Objekten zu kontrollieren, wodurch dem Betrachter das Verständnis der Szene erschwert wird.

Um dem Betrachter die Orientierung in der Augmented Reality Szene zu erleichtern, wurde die Textur eines Farbbildes, welches von einer vorgeschalteten Tiefenbildkamera generiert wurde, auf die aus zuvor akquirierten Bilddaten segmentierte Hautoberfläche des Patienten projiziert. Diese Patientenoberfläche wurde dann in das Farbbild der Tiefenbildkamera übertragen. So entsteht sozusagen eine künstliche Nachbildung des Körpers des Patienten, wodurch sich der Betrachter leichter in der virtuellen Szene zurechtfinden kann.

Zusätzlich wurde ein Shader implementiert, mit dessen Hilfe effizient zur Laufzeit entschieden werden kann, welche Pixel zu texturieren sind und welche nicht. Zu Letzterem gehören verdeckte Flächen, konkret also die Innenflächen des Körpers und Flächen, die von der Tiefenbildkamera, aus welcher das für die Textur genutzte Farbbild stammt, nicht sichtbar sind. Diese verdeckten Flächen sollen nicht texturiert werden, daher wurden die Innenflächen dunkel eingefärbt, um einen guten Kontrast zu den umgebenden Organen zu erhalten. Pixel, die von Tiefenbildkamera gesehen nicht sichtbar sind, werden verworfen und nicht gezeichnet.

Um einen Blick in das Innere des Patienten zu erlangen, wurde ein weiteres Shaderprogramm entwickelt, das innerhalb einer kreisförmigen Fläche die Ausgabe der virtuellen Patientenoberfläche unterdrückt. Je nach Position des mobilen Endgerätes wird diese Öffnung neu berechnet und hierdurch auf der Patientenoberfläche verschoben, sodass eine Betrachtung von verschiedenen Blickpunkten möglich ist.

Trotz der großen Anzahl an Rechenschritten, die binnen kürzester Zeit durchgeführt werden müssen, ist diese Implementierung echtzeitfähig und eignet sich somit gut für die Verwendung im Bereich der mobilen Augmented Reality. Durch die Kombination der Texturprojektion und der Öffnung wird dem Betrachter insgesamt die Orientierung in der virtuellen Szene erleichtert und zugleich sichergestellt, dass eine räumlich korrekte Einordnung der Organe möglich ist. So entsteht nicht mehr der Eindruck von oberhalb der Szene "schwebenden" Objekten, viel mehr lassen sich die Organe nun mental in der Gesamtszene einordnen.

Eine Gruppierung der Objekte in Fokus und lokaler und globaler Kontext ist nun möglich. Für diese wird jeweils eine individuelle Visualisierung umgesetzt. Über die Kategorisierung kann der Benutzer selbst entscheiden, welche Objekte tatsächlich visualisiert werden und worauf sein Hauptaugenmerk liegt. Hierdurch kann die Fülle des Bildes kontrolliert werden, wodurch ein überladenes Bild vermieden wird. Außerdem wird dadurch sichergestellt, dass die relevanten Strukturen im Bild hervorgehoben werden und der Betrachter somit nicht von anderen, weniger relevanten Objekten abgelenkt wird.

Da die Implementierung der Visualisierung unabhängig von der Logik der mobilen markerlosen Augmented Reality ist, lässt sich diese leicht in weitere Augmented Reality Anwendungen integrieren. Insbesondere eignet sie sich für weitere Projekte im Bereich der mobilen Augmented Reality, gerade weil dort eine Visualisierung in Echtzeit sehr viel wichtiger ist als in offline berechneten Augmented Reality Anwendungen.

Momentan wird in der Klassifizierung der Objekte zwischen drei Gruppen unterschieden. Eine feinere Untergliederung mit untereinander stärker divergierenden Visualisierungen wäre als Erweiterung denkbar. In diesem Zuge könnten zur Erreichung sinnvoller Effekte ebenfalls aufwendigere Bildbearbeitungsmechanismen umgesetzt werden, die den Umfang der vorliegenden Arbeit jedoch übersteigen würden. Eine komplette Umsetzung des Fokus-und-Kontext Frameworks, wie in [30] vorgestellt, in MITK ist denkbar und sinnvoll, was der zeitliche Rahmen dieser Thesis jedoch nicht zuließ. Im Allgemeinen bietet sich außerdem eine umfassende Evaluierung bei betroffenen Personengruppen an,

um die Güte der Visualisierung bewerten zu können. Diese sollte im Hinblick auf die Aussagekraft eine relativ hohe Anzahl an Teilnehmern haben, was zwangsläufig einen großen zeitlichen Aufwand für die Befragung und die Auswertung der gewonnenen Daten bedeutet. Auch die Generierung der Daten für die Befragung stellt ein Problem dar, da eine allgemeine Aussage über den Nutzen einer bestimmten Visualisierung ohne konkreten Anwendungsfall durchaus schwierig ist. Aus diesen Gründen wurde in dieser Thesis auf eine solche Evaluierung verzichtet.

Zusammenfassend ist im Rahmen dieser Arbeit eine Sammlung von Techniken zur Augmented Reality Visualisierung entstanden, die ohne großen Aufwand wiederverwendbar ist und durch eine echtzeitfähige Implementierung in vielerlei Anwendungen einsetzbar ist. Insbesondere die zu Beginn formulierten Ziele der Verbesserung der räumlichen Vorstellung von einer Augmented Reality Szene und der Vermeidung eines überfüllten Bildes konnten in dieser Arbeit umgesetzt werden.

7 REFERENZEN

[1] Statistisches Bundesamt, *Gesundheit, Todesursachen in Deutschland*, Fachserie 12 Reihe 4, Wiesbaden, 2013

[2] Internetauftritt des Projekts "Markerlose navigierte Chirurgie" der Medizinischen und Biologischen Abteilung des Deutschen Krebsforschungszentrums in Heidelberg, http://www.dkfz.de/de/mbi/research/MarkerloseChirurgie.html, zuletzt aufgerufen am 28. Februar 2014

[3] Wirht, C.-J.; *Komplikationen in Orthopädie und Unfallchirurgie: vermeiden, erkennen, behandeln*, Georg Thieme Verlag, 2010, S. 164

[4] Shuhaiber, J. H.; *Augmented reality in surgery*. Arch Surg, Department of Surgery, University of Illinois at Chicago, 2004, S. 170-174

[5] Müller, M.; *Augmented reality navigation for percutaneous neprholithotomy on mobile devices*, Dissertation, Universität Heidelberg, Medizinische Fakultät, Juli 2013

[6] Azuma, R. T. ; *A survey of augmented reality*, Presence, 1997, S. 355-385

[7] Wolf, I.; Vetter, M.; Wegner, I.; Nolden, M.; Böttger, T.; Hastenteufel, M.; Schöbinger, M.; Kunert, T.; Meinzer, H.-P.; Galloway, R. L.; *The Medical Imaging Interaction Toolkit (MITK) - a tool facilitating the creation of interactive software by extending VTK and ITK*, SPIE Medical Imaging 2004: Visualization, Image-Guided Procedures, and Display, 2004, S. 16-27

[8] Nolden, M.; Zelzer, S.; Seitel, A.; Wald, D.; Müller, M.; Franz, A. M.; Maleike, D.; Fangerau, M.; Baumhauer, M.; Maier-Hein, L.; Maier-Hein, K.; Meinzer, H. P.; Wolf, I., *The Medical Imaging Interaction Toolkit: challenges and advances,* International Journal of Computer Assisted Radiology and Surgery, Springer, 2013, S. 1-14

[9] Yung, K.; Seitel, A.; Mersmann, S.; Meinzer, H.-P. & Maier-Hein, L.; *MITK-ToF: Time-of-Flight Kamera-Integration in das Medical Imaging Interaction Toolkit,* Bildverarbeitung für die Medizin 2011, Springer, 2011, S. 204-208

[10] Seitel, A.; Yung, K.; Mersmann, S.; Kilgus, T.; Groch, A.; Santos, T. R. D.; Franz, A. M.; Nolden, M.; Meinzer, H.-P. & Maier-Hein, L., *MITK-ToF Range Data Within MITK,* International Journal of Computer Assisted Radiology and Surgery, 2011, S. 87-96

[11] Nischwitz, A.; Fischer, M.; Haberäcker, P.; Socher, G.; *Computergrafik und Bildverarbeitung, Band I: Computergrafik*, 3. Auflage, Vieweg + Teubner, S. 7-8

[12] Angel, E.; *Interactive Computer Graphics, A Top-Down Approach Using OpenGL*, Fifth Edition, Pearson Education, Inc., S. 403-407

[13] Nischwitz, A.; Fischer, M.; Haberäcker, P.; Socher, G.; *Computergrafik und Bildverarbeitung, Band I: Computergrafik*, 3. Auflage, Vieweg + Teubner, S. 122-124

[14] Shreiner, D.; Woo, M.; Neider, J.; Davis, T.; *OpenGL programming guide: the official guide to learning OpenGL*, version 2, Fifth Edition, Pearson Education, Inc., S. 2-4

[15] Nischwitz, A.; Fischer, M.; Haberäcker, P.; Socher, G.; *Computergrafik und Bildverarbeitung, Band I: Computergrafik*, 3. Auflage, Vieweg + Teubner, S. 46-50

[16] Shreiner, D.; Woo, M.; Neider, J.; Davis, T.; *OpenGL programming guide: the official guide to learning OpenGL*, version 2, Fifth Edition, Pearson Education, Inc., S. 645-663

[17] Schroeder, W.; Martin, K.; Lorensen B.; *The visualization toolkit: an object-oriented approach to 3D graphics*, Prentice Hall PTR, 1996, S. 6-93

[18] Ayachit, U.; Cole, D.; Geveci, B.; Hoffman, B.; Ibanes, L.; Jomier, J.; Mar-

tin, K.; Squillacote, A.; *Kitware's Software Developer's Quarterly – VTK Shaders*, Issue 2, Oct 2006, Kitware Inc., Clifton Park, New York

[19] Salvi, J.; Pagès, J.; Batlle, J.; *Pattern Codification Strategies in Structured Light Systems.,* Pattern Recognition, Volume 37, Issue 4, April 2004, S. 827–849

[20] Lange, R.; *3D Time-of-Flight Distance Measurement with Custom Solid-State Image Sensors in CMOS/CCD-Technology*, University of Siegen, 2000

[21] Maier-Hein, L.; Franz, A. M.; Fangerau, M.; Schmidt, M.; Seitel, A.; Mersmann, S.; Kilgus, T.; Groch, A.; Yung, K.; dos Santos, T. R. & Meinzer, H.-P.; *Towards Mobile Augmented Reality for On-Patient Visualization of Medical Images*, Bildverarbeitung für die Medizin, Springer, 2011, S. 389-393

[22] Müller, M.; Rassweiler, M.-C.; Klein, J.; Seitel, A.; Gondan, M.; Baumhauer, M.; Teber, D.; Rassweiler, J.; Meinzer, H.-P. & Maier-Hein, L.; *Mobile augmented reality for computer-assisted percutaneous nephrolithotomy,* International Journal of Computer Assisted Radiology and Surgery, Springer Berlin Heidelberg, 2013, S. 663-675

[23] dos Santos, T. R.; Seitel, A.; Meinzer, H.-P. & Maier-Hein, L.; *Correspondences Search for Surface-Based Intra-Operative Registration,* Medical Image Computing and Computer-Assisted Interventions (MICCAI), Springer, 2010, S. 660-667

[24] Besl, P. J. & McKay, N. D. A; *Method for Registration of 3-D Shapes*, IEEE T Pattern Anal, 1992, 14, S. 239-256

[25] Horn, B. K. P.*; Closed-form solution of absolute orientation using unit quaternions,* J. Opt. Soc. Am. A, OSA, 1987, S. 629-642

[26] Maier-Hein, L.; Franz, A.; dos Santos, T.; Schmidt, M.; Fangerau, M.; Meinzer, H.-P. & Fitzpatrick, J. M.; *Convergent Iterative Closest-Point Algorithm to Accomodate Anisotropic and Inhomogenous Localization*

Error, IEEE T Pattern Anal, IEEE, 2012, 34, S. 1520-1532

[27] Heim, E.; Kilgus, T.; Haase, S.; Iszatt, J.; Franz, A.; Müller, M.; Fangerau, M.; Hornegger, J.; Meinzer, H.-P. & Maier-Hein, L; *GPGPU-beschleunigter anisotroper ICP zur Registrierung von Tiefendaten*, Bildverarbeitung für die Medizin, Springer, 2014, to appear

[28] Hansen, C.; Wieferich, J.; Ritter, F.; Rieder, C. & Peitgen, H.-O.; *Illustrative visualization of 3D planning models for augmented reality in liver surgery,* International Journal of Computer Assisted Radiology and Surgery, Springer-Verlag, 2010, 5, S. 133-141

[29] Lerotic, M.; Chung, A. J.; Mylonas, G. P. & Yang, G.-Z.; *pq-space Based Non-Photorealistic Rendering for Augmented Reality*, Medical Image Computing and Computer-Assisted Interventions (MICCAI), 2007, S. 102-109

[30] Kalkofen, D.; Mendez, E. & Schmalstieg, D., *Comprehensible Visualization for Augmented Reality Visualization and Computer Graphics*, IEEE Transactions on, 2009, 15, S. 193-204

[31] Hering, J.; Gergel, I.; Krömker, S.; Meinzer, H.-P. & Wegner, I. Handels, H.; Ehrhardt, J.; Deserno, T. M.; Meinzer, H.-P. & Tolxdorff, T.; *MITK-OpenCL: Eine Erweiterung für das Medical Imaging Interaction Toolkit,* Bildverarbeitung für die Medizin, Springer, Berlin, 2011, S. 254-258

II. ANHANG

A ABKÜRZUNGSVERZEICHNIS

CPU	Central Processing Unit
CT	Computertomographie
GLSL	Open Graphics Library Shading Language
GPU	Graphics Processing Unit
GUI	Grafische Benutzeroberfläche
MITK	The Medical Interaction Toolkit
MRT	Magnetresonanztomographie
ToF	Time-Of-Flight
ToFoscopy	Markerlose, mobile Augmented Reality
VTK	The Visualization Toolkit

B ABBILDUNGSVERZEICHNIS

Abbildung 1: Bisherige Augmented Reality Visualisierung mit geringer Tiefenwahrnehmung (siehe Tumor) *4*

Abbildung 2: Überblick über durchzuführende Transformationen, um Objektkoordinaten in Bildschirmkoordinaten zu überführen. Die Modell- und Augenpunktstransformation (rotes Rechteck) dient zur Berechnung der Weltkoordinaten. Eine Projektionstransformation (gelbes Rechteck) legt das sichtbare Volumen (Frustum) der Szene fest. Nach einer Normierung werden mithilfe einer Viewport-Transformation (grünes Rechteck) die Bildschirmkoordinaten berechnet. Nach: [13] *9*

Abbildung 3: Ablauf der Bearbeitung eines Eingangsvertex durch einen Vertex- und einen Fragmentshader. Die Eingabewerte (grünes Rechteck) des Vertex und Werte aus der Anwendung, die den Shader aufruft ("`uniform`", blaues Rechteck), werden im Vertex-Shader verarbeitet. Die transformierte Position des Vertex (`gl_Position`, oranges Rechteck) wird zur Weiterverarbeitung in der Pipeline ausgegeben. Als Eingabeparameter erhält der Fragment-Shader verschiedene eingebaute und benutzerdefinierte Attribute aus dem Vertex-Shader ("`varying`, lila Pfeil) und Werte aus der Anwendung ("`uniform`", blaues Rechteck), die den Shader aufruft. Als Ausgabeparameter wird die endgültige Farbe eines Fragments (`gl_FragColor`, rotes Rechteck) in den Bildschirmspeicher geschrieben. Nach [16] *11*

Abbildung 4: VTK Rendering Pipeline: Eingabedaten (Source) werden zunächst gefiltert (Filter) und zu visualisierbaren Objekten konvertiert (Mapper). Diese Objekte nennen sich Actor (blau). Einem Actor werden verschiedene Visualisierungsparameter hinzugefügt, bis schließlich die perspektivische Darstellung auf dem Bildschirm vom Renderer und zugehörigen RenderWindows realisiert wird. Nach [17] *12*

Abbildung 5: Struktur des Moduls MITK-ToF, bestehend aus drei Schichten: Die Hardwareschicht ToFHardware (grün), die Verarbeitungsschickt ToFProcessing (rot) und die Anwendungsschicht ToFUI (gelb). Diese Schichten stellen die Schnittstelle zur Kamerahardware (ToFHardware) und zum Benutzer (ToFUI) her und sind weiterhin zuständig für die Verarbeitung der gewonnenen Bilddaten (ToFProcessing). MITK-ToF kann beliebig von verschiedenen Bundles genutzt werden, was exemplarisch im hellroten Bereich "MITK" dargestellt ist. Quelle: [10] *14*

Abbildung 6: Prinzip der ToFoscopy. Einem mobilen Endgerät wird eine Tiefenbildkamera vorgeschaltet. Das Distanzbild aus der Tiefenbildkamera wird zur Registrierung mit der segmentierten Hautoberfläche des Patienten aus zuvor aufgenommenen Bilddaten (CT/MRT) genutzt. Auf dem mobilen Endgerät kann das Farbbild der realen Szene in Kombination mit Augmented Reality Objekten, die ebenfalls aus dem zuvor akquirierten Bilddatensatz gewonnen werden, betrachtet werden. Mithilfe der Registrierung können die virtuellen Objekte relativ zur aktuellen Kameraposition transformiert werden. Quelle: [21] *17*

Abbildung 7: Vergleich konventioneller Augmented Reality Visualisierung (links) mit der Visualisierung von Objekten als Kontur (rechts) aus [28]. Verschiedene Strichstärken der Umrisslinien (grüner Rahmen, Strichstärke steigt von links nach rechts) und unterschiedlich stark

schraffierte Oberflächen (roter Rahmen, linker Ast wird mit dickeren Linien schraffiert als rechter Ast) sollen eine bessere räumliche Vorstellung der Szene bewirken. Quelle: [28] *19*

Abbildung 8: Aus der Originalszene (links) wird eine Oberfäche für die Verwendung für Augmented Reality extrahiert und rekonstruiert (rechts). Zur Orientierung des Betrachters beitragende Elemente der Originalszene (hier in hellerem blau dargestellte Kanten) werden beibehalten. Quelle: [29] *20*

Abbildung 9: Fusion einer virtuellen Oberfläche, die aus der Originalszene rekonstruiert wurde (blau) und virtuellem Objekt (rosa). Eine Erhaltung der Kanten der Originalszene in der virtuellen Oberfläche (blau) erleichtert das Verständnis der Szene. Quelle: [29] *20*

Abbildung 10: Fokus-und-Kontext-Unterteilung am Beispiel eines Autos. Zusammenspiel von Elementen aus der realen Szene (Karosserie, als Kontext definiert) und virtuellen, in der realen Szene unsichtbaren Elementen (Reifen, Motor, als Fokus definiert), um zur Orientierung beitragende Strukturen der Originalszene zu erhalten statt durch virtuelle Objekte zu verdecken. Quelle: [30] *21*

Abbildung 11: Vorgehen zur Umsetzung der definierten Ziele *23*

Abbildung 12: QmitkToFVisualization (grün) innerhalb der MITK Architektur. Das in dieser Thesis entwickelte Widget QmitkToFVisualization ist in die Anwendungsschicht von MITK-ToF integriert. Dort befinden sich ebenfalls zwei entwickelte Shaderprogramme (textureShader und holeShader), die sowohl getrennt als auch gemeinsam (AR Shader) nutzbar sind. *25*

Abbildung 13: Ausgangssituation für die Berechnung von Texturkoordinaten für ein bestimmtes Objekt (blauer Würfel) mithilfe der Klasse `vtkProjectedTexture`*. Eine Kamera nimmt ein Farbbild (rotes Trapez) auf. Auf den im Würfel eingezeichneten Punkt soll der Punkt des Farbbildes abgebildet werden, der auf der Geraden zwischen Objektpunkt und Kamera liegt (orange). Das Koordinatensystem wird von VTK zur Beschreibung der Lage aller abgebildeter Objekte verwendet.* *27*

Abbildung 14: Durch die verschobene Kameraposition und –orientierung muss auf den Objektpunkt (hier auf der orangen Gerade) aus Abbildung 13 ein anderer Bildpunkt des Farbbildes gemappt werden. *28*

Abbildung 15: Intervalle der Texturkoordinaten in x-Richtung ("SRange" in rot) und y-Richtung ("TRange" in blau) der Textur (exemplarisch als schwarzes Rechteck abgebildet). *29*

Abbildung 16: Verschobene Texturprojektion, entstanden durch Fertigungstoleranzen der Kamera. *29*

Abbildung 17: Projektion der Information aus einem Farbbild (links oben) auf eine aus zuvor akquirierten Bilddaten segmentierte Patientenoberfläche (links unten). Ergebnis ist eine Fusion der Farbinformation und der Patientenoberfläche (rechts). *31*

Abbildung 18: Problematik der Texturprojektion mit der `vtkProjectedTexture`*. Mehreren Objektpunkten (roter und grüner Punkt) wird dieselbe Texturkoordinate zugewiesen, wenn sie sich von der Tiefenbildkamera gesehen auf einer Geraden (orange in dieser Abbildung) befinden. Resultierend daraus wird die Textur des Farbbildes rundum auf das Objekt projiziert.* *32*

Abbildung 19: Vertex-Shader zur perspektivischen Transformation des Eingangsvertex von Objektkoordinaten zu Projektionskoordinaten. 33
Abbildung 20: Texturierung im Fragment-Shader des Shaderprogramms zur Texturprojektion 33
Abbildung 21: Verwerfen nicht sichtbarer Pixel im Fragment-Shader. Mithilfe der Koordinate des Ausgangsvertex wird überprüft, ob die Normale des Ausgangsvertex mit dem Verbindungsvektor zwischen dem Vertex und der Position der Tiefenbildkamera einen Winkel größer 90 Grad einschließen. Wenn ja, wird das zugehörige Fragment verworfen. 34
Abbildung 22: Skizze zur Berechnung der Fragmente, die von der Position der Tiefenbildkamera („Kameraposition", blau) nicht sichtbar sind, und somit verworfen werden sollen. Die zum Vertex gehörige Normale schließt mit dem Verbindungsvektor d (grün) einen Winkel von weniger als 90 Grad ein. Das Fragment gehört somit zu den von der Tiefenbildkamera sichtbaren Fragmenten und soll daher nicht verworfen werden. 35
Abbildung 23: Selbige Berechnung wie Abbildung 22 mit einem Vertex, der von der Tiefenbildkamera gesehen nicht sichtbar ist. Der Verbindungsvektor d schließt mit der Normale einen Winkel von mehr als 90 Grad ein. Deshalb soll das Fragment verworfen werden. 36
Abbildung 24: Vorher: Farbinformation ist nicht entsprechend der Originalszene nur auf dem Teil der Oberfläche zu sehen, der von der Kamera erfasst werden kann (hier die Vorderseite der Patientenoberfläche (links)), sondern ebenfalls auf der Rückseite der Patientenoberfläche (rechts). 36
Abbildung 25: Nachher: Optimierte Texturprojektion mithilfe des Shaderprogramms, sodass die Textur des Farbbildes aus der Tiefenbildkamera nur auf die Vorderseite der Patientenoberfläche (links) projiziert wird. Im rechten Bild blickt der Betrachter auf die Rückseite des Körpers. Durch das Verwerfen der Front-Facing Fragmente wird die Rückseite durchsichtig, sodass man nur die Innenfärbung des Körpers erkennt. 37
Abbildung 26: Front-Facing Fragmente, die von der Tiefenbildkamera nicht sichtbar sind, werden durchsichtig, sodass nur die dahinterliegenden Fragmente (also die Innenseiten des Körpers) sichtbar sind. 38
Abbildung 27: Färbung des Innenraums des Körpers im Fragment-Shader, um einen guten Kontrast zu den darin befindlichen Organen zu erhalten 38
Abbildung 28: Auszug aus der Datei files.cmake zum automatischen Laden des Shaders 39
Abbildung 29: Zugriff auf den Shader innerhalb des Programms, das den Shader aufruft 40
Abbildung 30: Vertex-Shader zur perspektivischen Transformation des Eingangsvertex von Objektkoordinaten zu Projektionskoordinaten. 40
Abbildung 31: Fragment-Shader: Berechnung einer Öffnung in die Hautoberfläche des Patienten um eine verbesserte Tiefenwahrnehmung zu erreichen 41
Abbildung 32: Projektion einer Öffnung in eine aus Bilddaten segmentierte Oberfläche mithilfe eines Shaderprogramms, um eine Tiefenwahrnehmung in der Szene zu vermitteln. Die Öffnung verschiebt sich je nach Lage der Oberfläche. 41
Abbildung 33: GUI zur Gruppierung der in der Augmented Reality Szene dargestellten Objekte. Fokusstrukturen sind solche, auf denen das Hauptaugenmerk liegt. Als Local Context sind

umliegende Objekte der Fokustruktur definiert und Global Context Objekte sind solche, die weiter entfernt von der Fokusstruktur liegen. *44*
Abbildung 34: Vorher: Bisherige Augmented Reality Visualisierung ohne Gruppierung. Hier ist nicht erkennbar, welche Strukturen für den Betrachter und den Anwendungsfall relevant sind. *45*
Abbildung 35: Nachher: Die optische Unterteilung zwischen Fokus (grün), lokalem Kontext (rot) und globalem Kontext (weiß) hilft, das Hauptaugenmerk des Betrachters auf eine bestimmte Struktur zu lenken, sodass dieser relevante Strukturen schnell erkennt und von anderen umliegenden Strukturen nicht abgelenkt wird. *45*
Abbildung 36: Auswahlmöglichkeit über das GUI, Gruppen zu verbergen *46*
Abbildung 37: Visualisierung von Fokus (grün) und lokalem Kontext (rot) ohne globalen Kontext. Das Bild wirkt hierdurch weniger überladen und eine Beschränkung auf die wesentlichen Strukturen wird ermöglicht. *46*
Abbildung 38: Vorher: Bisherige Visualisierung der Patientenoberfläche im Farbbild. Das virtuelle Objekt verdeckt Informationen aus der realen Szene, sodass Informationen daraus verloren gehen. *47*
Abbildung 39: Nachher: Unten: Einblendung des virtuellen, texturierten Körpers (oben links) in das Farbbild (oben rechts). Statt das Farbbild zu überlagern, werden Informationen der realen Szene aufgegriffen und mit dem virtuellen Körper so kombiniert, dass dieser dem Körper in der realen Szene entspricht. *48*
Abbildung 40: Projektion einer Öffnung auf die aus zuvor aufgenommenen Bilddaten segmentierte Hautoberfläche des Patienten zur Verbesserung der Tiefenwahrnehmung. *49*
Abbildung 41: Berechnung der Öffnung in die CT-Hautoberfläche des Patienten, um die Tiefenwahrnehmung in der Augmented Reality Szene zu verbessern. Der Radius ist flexibel einstellbar. Die Position der Öffnung passt sich der Blickrichtung des Betrachters an. *49*
Abbildung 42: Links: Vorher: Augmented Reality mit geringer Tiefenwahrnehmung, sodass der Eindruck entsteht, dass die Augmented Reality Objekte über der Haut schweben. Rechts: Nachher: Verbesserte Tiefenwahrnehmung in der Augmented Reality Szene durch die Kombination aus einer Öffnung und einer Oberfläche. Diese Kombination ermöglicht die räumlich korrekte Einordnung der Organe. *50*
Abbildung 43: Vorher: Mobile Augmented Reality mit geringer Tiefenwahrnehmung, sodass die virutellen Objekte als oberhalb der Szene platziert wirken. *51*
Abbildung 44: Nachher: Mobile Augmented Reality mit verbesserter Tiefenwahrnehmung. Durch eine projizierte Textur auf die Hautoberfläche des Patienten wird die Orientierung in der Augmented Reality Szene erleichtert. Eine Öffnung in die Hautoberfläche ermöglicht eine korrekte räumliche Einordnung der Organe. Die Anwendung des Fokus-und-Kontext Konzepts führt zu einer optischen Unterteilung der Organe in verschiedene Gruppen. *52*

C TABELLENVERZEICHNIS

Tabelle 1: Überblick über wichtige Attribute und Variablen eines Vertex- und eines Fragment-Shaders. Nach [16] *10*

D Ausgewählte Implementierungen

1. *textureShader.xml:*

```
<?xml version='1.0' encoding='UTF-8'?>
 <Material name='GenericAttributes1'>
 <Shader scope='Vertex' name='VertexShader' location='Inline'
            language='GLSL' entry='main'>
  varying vec3  vertex;
  varying vec3  normal;
  varying vec3  vertexWorldCoords;
  varying vec3  normalWorldCoords;
  void main(void)
  {
   vertexWorldCoords = vec3(gl_ModelMatrix * gl_Vertex);
   normalWorldCoords = normalize(mat3(gl_ModelMatrix) *
                                  gl_Normal);
   vertex = vec3(gl_ModelViewMatrix * gl_Vertex);
   normal = normalize(gl_NormalMatrix * gl_Normal);
   gl_TexCoord[0]  = gl_MultiTexCoord0;
   gl_Position     = ftransform();
  }
 </Shader>
 <Shader scope='Fragment' name='FragmentShader' location='Inline'
            language='GLSL' entry='main'>
  varying vec3  vertex;
  varying vec3  normal;
  varying vec3  vertexWorldCoords;
  varying vec3  normalWorldCoords;
  uniform vec3 projectorPos;
  uniform sampler2D texSampler;
  void main(void)
```

```
29    {
30     vec3 tnorm = normal;
31     vec4 textureColor;
32     vec4 finalColor;
33     if(gl_FrontFacing)
34     {
35      textureColor = texture2D(texSampler, vec2(gl_TexCoord[0]));
36      finalColor = textureColor;
37     vec3 d = normalize(projectorPos - vertexWorldCoords);
38     float scalar = d.x*normalWorldCoords.x + d.y *
39               normalWorldCoords.y + d.z*normalWorldCoords.z;
40     if ((scalar < 0.0))
41     {
42      discard;
43     }
44    }
45   else
46    {
47     float f = 1-abs(tnorm.z);
48     f=f/2;
49     finalColor = vec4(f,f,f,1.0);
50     }
51    gl_FragColor = finalColor;
52    }
53   <ApplicationUniform type="float" name="HoleSize"
54               number_of_elements="1" > </ApplicationUniform>
55   </Shader>
56  </Material>
```

2. *holeShader.xml:*

```
<?xml version='1.0' encoding='UTF-8'?>
 <Material name='GenericAttributes1'>
 <Shader scope='Vertex' name='VertexShader' location='Inline'
             language='GLSL' entry='main'>
  varying vec3  vertex;
  varying vec3  normal;
  void main(void)
  {
   vertex = vec3(gl_ModelViewMatrix * gl_Vertex);
   normal = normalize(gl_NormalMatrix * gl_Normal);
   gl_Position     = ftransform();
  }
 </Shader>
 <Shader scope='Fragment' name='FragmentShader' location='Inline'
             language='GLSL' entry='main'>
 <Uniform name="LightPosition" type="vec3" number_of_elements="3"
             value="0.0 1000.0 0.0"></Uniform>
 <Uniform name="SkyColor" type="vec3" number_of_elements="3"
             value="0.9 0.8 0.5"></Uniform>
<Uniform name="GroundColor" type="vec3" number_of_elements="3"
             value="0.1 0.1 0.1"></Uniform>
  varying vec3  vertex;
  varying vec3  normal;
  uniform vec3 LightPosition;
  uniform vec3 SkyColor;
  uniform vec3 GroundColor;
  uniform float HoleSize;
  void main(void)
  {
```

```
30      vec3 tnorm = normal;
31      vec4 finalColor;
32      if(gl_FrontFacing)
33      {
34       float distanceZAxis = sqrt(dot(vertex.xy,
35                                      vertex.xy));
36       if(distanceZAxis < HoleSize )
37       {
38        discard;
39       }
40       if(abs(HoleSize - distanceZAxis) < 2.5)
41       {
42       finalColor = vec4(0.97,0.83,0.35,1.0);
43       }
44      }
45      else
46      {
47       float f = 1-abs(tnorm.z);
48       f=f/2;
49       finalColor = vec4(f,f,f,1.0);
50      }
51     gl_FragColor = finalColor;
52     }
53    <ApplicationUniform type="float" name="HoleSize"
54                number_of_elements="1" > </ApplicationUniform>
55    </Shader>
56  </Material>
```

E DVD

Die angefügte DVD enthält zum einen diese Arbeit in elektronischer Form und zum anderen ein Video zur Demonstration der mobilen markerlosen Augmented Reality mit der in dieser Arbeit entwickelten Visualisierung.

III. EIDESSTATTLICHE ERKLÄRUNG

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Bachelorarbeit

Augmented Reality Visualisierung medizinischer Bilddaten auf mobilen Geräten

selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Die Stellen der Arbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellung sowie für Quellen aus dem Internet.

Heilbronn, März 2014

Tamara Wiebe